CERFACS
CFD - COMBUSTION

Sous la direction Laurent Gicquel et Yannick Sommerer

OPTIMISATION DE TURBOMOTEUR D'HELICOPTERE - STAGE EN COLLABORATION AVEC TURBOMECA

Rapport de stage de Mastère
Mécanique des Fluides Numérique

Florent DUCHAINE

Septembre 2004

Ref : WN/CFD/04/76

Département Hydraulique et Mécanique des fluides
ENSEEIHT

Remerciements

La réalisation d'un projet dans l'équipe CFD du CERFACS est liée à une effervescence collective qui m'invite dès le debut de ces remerciements à m'escuser auprès des personnes qui ne sont pas citées.

En premier, je tiens à exprimer ma gratitude à Laurent Gicquel et à Yannick Sommerer qui m'ont encadré durant ce stage, me laissant la liberté d'agir tout en m'apportant le nécessaire pour que mes travaux aboutissent. Ensuite, je remercie Thierry Morel et Samuel Buis pour leur aide précieuse et leur conseils avisés pour l'utilisation de PALM.

Je transmets mes remerciements à toute l'équipe CFD pour leur disponibilité, leurs conseils et leur bonne humeur. Mes pensées vont plus particulièrement vers Valérie Auffray et mes collègues de bureau Lea Artal et Bertrand Varoquié qui auront eu, entre autre, le mérite de me supporter.

Je n'oublie pas le grand maître des lieux, Thierry Poinsot, pour l'attention qu'il sait donner à chacun et pour la motivation qu'il transmet à l'équipe.

Enfin, je réserve une pensée toute particulière à Mélanie pour son soutien et sa patience.

Résumé

Le couplage entre l'optimisation et le calcul scientifique en Mécanique des Fluides aboutit à l'élaboration d'outils d'aide à la décision très puissants. C'est dans le cadre du projet Européen INTELLECT D.M. que TURBOMECA et le CERFACS se sont lancés dans une telle entreprise dont le but est de maitriser la température en sortie de chambre de combustion.

Ce rapport présente tout d'abord les résultats de l'étude bibliographique menée sur l'optimisation ainsi que sur son utilisation couplée avec des codes de calculs. Les premiers pas vers la constitution d'une boucle automatique d'optimisation sont ensuite abordés en étudiant les possibilités d'intégration du code de Mécanique des Fluides N3SNatur dans un algorithme du simplexe. Enfin, des tests sur une configuration de base permettant de valider l'outils réalisé sont exposés.

Abstract

Coupling optimization and Computation Fluid Dynamics codes leads to the realisation of very powerful tools. TURBOMECA and CERFACS, in the context of the European project INTELLECT D.M., begin to develop an automatic chain optimization to satisfy pre-requisites on temperature profile at the outlet of a combustion chamber.

This report present first a bibliography on optimization and its use coupling with a calculation code. Then, the first steps of the developement of a close loop optimization based on the RANS calculation code N3SNatur integrated in a simplex alogrithm are presented. Finally, tests which permit to validate the tool are explained.

Mots clefs

Optimisation, Mécanique des Fluides Numérique, algorithme du simplexe, N3SNatur, PALM

Table des matières

Introduction

L'optimisation, ça peut rapporter gros !!! C'est sur ces mots de J.NOAILLES (ENSEEIHT-IRIT) qu'a débuté la journée optimisation organisée à l'ENSEEIHT (Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications, Toulouse 31) le 4 Mars 2004. Les techniques d'optimisation sont aujourd'hui très largement utilisées dans de nombreux domaines. De ce fait, il existe de multiples méthodes plus ou moins performantes selon le cas étudié et la principale difficulté est de trouver celle qui sera la plus adaptée au problème traité. L'approfondissement de ces techniques appliquées à un problème spécifique doit permettre un gain important devant l'investissement qu'il engendre.

L'étude présentée dans ce rapport a été réalisée au CERFACS dans le cadre du projet européen INTELLECT D.M. dont le but est la mise en oeuvre de méthodologies pour le design de chambres de combustion à faible émission de polluant. Elle reflète des besoins de l'industriel et partenaire TURBOMECA et concerne plus particulièrement la maitrise de la température en sortie d'une chambre de combustion d'hélicoptère. En effet, en sortie de chambre se trouve une turbine dont les pâles se détériorent en cas de forte chaleur des gaz incidents. Actuellement, les travaux reposent essentiellement sur les connaissances et le savoir faire des ingénieurs : l'optimisation est réalisée "à la main" par le post-processing de chaque run qui donne intuitivement une nouvelle configuration à tester. L'utilisation de l'optimisation automatique couplée à la CFD possède deux finalités principales : d'une part, elle doit permettre de répondre rapidement et précisément au problème de l'usure des turbines, et d'autre part, enrichir le potentiel scientifique des personnes travaillant sur ce type de moteur.

Le travail consiste à développer une chaîne d'optimisation basée sur le code de *CFD* (Computational Fluid Dynamic) N3SNatur (SIMULOG). Le but du projet est de mettre en oeuvre un outil puissant d'aide à la conception qui soit : automatique, robuste, portable, permettant de prendre en compte plusieurs critères (optimisation multi-critère ou multi-objectif) et des contraintes.

Ce travail entre en compte dans la formation de Mastère en Mécanique des Fluides Numérique de l'ENSEEIHT en tant que Stage-projet. Etant donné l'intérêt porté au sujet de ce stage et le désir de continuer à travailler dans l'équipe *Combustion* du CERFACS, une poursuite en thèse aura lieu en octobre 2004.

Le présent rapport se compose de quatre parties. Tout d'abord, le contexte de l'étude est présenté. Ensuite, une vue très générale des techniques d'optimisation est proposée avec les spécificités apportées par leur utilisation dans le cadre de la *CFD*. Puis, le travail d'intégration du code N3SNatur dans une boucle d'optimisation sera traité. Enfin, une partie des calculs sur des configurations de base, servant à tester l'outil, sera détaillée.

Chapitre 1

Contexte de l'étude

1.1 Le projet européen INTELLECT D.M.

Le projet européen dans lequel s'inscrit ce stage porte l'acronyme INTELLECT D.M. comme : INTEgrated Lean Low Emission CombusTor Design Methodology. Il regroupe plusieurs partenaires industriels désireux de voir une problématique de recherche et développement se solutionner par une effervescence collective. On compte parmi ces industriels Rolls Royce, SNECMA Moteur, TURBOMECA, l'ONERA, le CERFACS ...

L'objectif repose sur un fait hautement d'actualité qui est la pollution. En effet, la course actuelle des motoristes est basée sur la réduction des émissions de polluant et principalement des oxydes d'azote, couramment appelés NOx. Des normes de plus en plus exigentes sur la qualité de l'environnement poussent les industriels concernés à faire de gros efforts pour améliorer la conception des chambres de combustion.

Ce projet doit permettre d'établir des principes et des technologies nouvelles pour la conception de chambres de combustion, qui permettront de rendre les entreprises européennes compétitives tant au niveau performance que du contrôle de la pollution. Parmi les sujets abordés, on trouve le refroidissement, l'allumage, le réallumage en altitude, l'optimisation de l'utilisation d'air dans la turbine.

Le CERFACS est engagé à deux niveaux dans ce programme :

- une première tâche consiste à effectuer des simulations (directes et aux grandes échelles) pour comprendre l'aérodynamique et la thermique de l'écoulement au niveau d'une plaque multiperforée, en vue de l'écriture de lois de parois implémentable dans des codes RANS (Reynolds Average Navier-Stokes) industriels. (Travail de S. Mendez)
- la seconde, object de ce rapport, consiste à l'établissement d'une chaîne automatique d'optimisation construite autour d'un code RANS

C'est en collaboration avec TURBOMECA que le CERFACS doit mettre en place une chaîne d'optimisation automatique en vue d'un calcul sur un turbomoteur d'hélicoptère.

1.2 TURBOMECA

TURBOMECA fait partie depuis 2000 du groupe SNECMA qui figure parmi les quatre principaux motoristes mondiaux, spécialisé dans la propulsion, les équipements et les services associés. Avec une large gamme de systèmes de propulsion, SNECMA travaille pour la conception d'appareils aussi bien civils que militaires pour des sociétés telles que Airbus, Arianespace, Boeing, Dassault Aviation et Eurocopter.

TURBOMECA, société anonyme fondée en 1938, est le leader mondial de la conception, la production et la vente de turbines à gaz de petites et moyennes puissances pour hélicoptères. Elle fabrique également des turboréacteurs pour avions et missiles, ainsi que pour différentes applications terrestres et marines. Son succès résulte d'un investissement permanent en Recherche & Développement qui lui permet de motoriser les plus célèbres référence du marché : Eurocopter, Sikorsky, Agusta, Denel, Kamov, Boeing etc.

La mission de TURBOMECA concernant la tâche d'optimisation est de fournir la configuration de référence (programme LOPOCOTEP - LOw POlluant COmbustor TEchnology Programme) permettant de tester l'application réalisée ainsi que de valider les procédures de tests et les résultats obtenus.

1.3 Le CERFACS : Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique

Le CERFACS, crée en 1987, est un centre de recherche dont le but est de développer des méthodes avancées pour la simulation numérique et les algorithmes en vue de la résolution de problèmes technologiques aussi bien dans le domaine de la recherche que de l'industrie.

Le CERFACS est dirigé par J.C. Andre. Cinq entreprises se partagent son capital : le CNES (Centre National d'Études Spatiales), EADS France (European Aeronautic Defence and Space Company), EDF (Électricité De France), Météo France et la SNECMA.

Il héberge une centaine de personnes parmi lesquelles on compte des chercheurs et ingénieurs permanents mais aussi beaucoup de stagiaires, doctorants et post-doctorants de diverses nationalités. Ils se repartissent en cinq équipes :
- Parallel Algorithms
- Computational Fluid Dynamics (CFD)
- Climate Modelling and Global Change
- Electromagnetism
- Image and Signal Processing

L'équipe CFD se divise en deux pôles de recherche : Aérodynamique et Combustion. Ce stage s'est déroulé au sein de l'équipe Combustion (dirigée par T. Poinsot) dont les principales activités sont :
- la Simulation Numérique Directe pour l'étude des intéractions flamme/turbulence avec chimie complexe
- la Simulation des Grandes Echelles pour étudier le mélange, les instabilités de combustion et les interactions flamme/paroi en combustion gazeuse et diphasique gaz-liquide

La plupart des projets de l'équipe sont réalisés à partir du code AVBP développé au CERFACS. Le projet INTELLECT D.M. voit le retour de l'utilisation de code RANS (Reynolds Average Navier-Stokes) dans l'équipe avec l'utilisation de N3SNatur. Ce code, utilisé en tant que code de production en combustion par SNECMA, est développé par SNECMA, RENAULT, EDF et INKA/SIMULOG qui en est le maître d'oeuvre.

Les moyens de calcul disponibles au CERFACS sont :
- SGI Origin 2000, 32 processeurs
- COMPAQ AlphaServer SC, 40 processeurs
- PC Cluster, 16 processeurs
- De nombreuses stations de travail sur le réseau local

Chapitre 2

L'optimisation et la mécanique des fluides

Les problèmes d'optimisation occupent aujourd'hui une grande place dans la communauté scientifique. Ils ont toujours connu cette notoriété mais celle-ci se retrouve amplifiée par l'essor actuel des techniques informatiques. Le monde réel offre une multitude de problèmes d'optimisation dans des domaines très variés, tout pouvant être prétexte à de l'optimisation. De ce fait, il existe de nombreuses techniques en dépendance directe avec le type de problèmes qui peuvent être entre autres :

- combinatoires ou à variables continues
- à un ou plusieurs objectifs
- avec ou sans contraintes

Dans ce chapitre, le vocabulaire de base utilisé dans le rapport est présenté ainsi qu'une vue d'ensemble des techniques d'optimisation. Ensuite, le problème du couplage entre CFD et optimisation est abordé. Enfin, le simplexe, algorithme retenu pour tester la faisabilité du projet, sera introduit.

2.1 Définitions de base

Dans le but de clarifier les termes utilisés dans ce rapport, les *définitions* suivantes sont exposées[1] :

- Problème d'optimisation
- Espace d'état/recherche
- Variables/paramètres d'optimisation
- Fonction objectif/coût
- Minimum/Maximum local et global
- Bassin d'attraction
- Contrainte
- Méthode d'optimisation

Un problème d'optimisation est défini par un espace d'état, une ou plusieurs fonctions objectif et un ensemble (potentiellement vide) de contraintes.

L'espace d'état/de recherche est défini par l'ensemble des domaines de définition des variables d'optimisation du problème. Dans la plupart des problèmes, cet espace est fini car lié à une réalité physique (ce qui permet aussi de restreindre l'espace de recherche de l'algorithme et par conséquent, le temps de calcul). On appellera **taille du problème** la dimension de l'espace

[1] Pour un contexte plus formel concernant le vocabulaire de l'optimisation, le lecteur pourra se reporter à [3]

de recherche (nombre de variables d'optimisation).

Les variables/paramètres d'optimisation sont les quantités qui vont évoluer au cours du processus d'optimisation et faire varier la valeur de la fonction objectif. Elles peuvent être (dans un même problème) de nature diverse (réelle, entière, booléenne ...) et exprimer des données qualitatives ou quantitatives.

La fonction objectif/coût est une paramétrisation du but que l'on souhaite atteindre. L'algorithme d'optimisation cherche les variables qui permettent d'obtenir un minimum ou un maximum de cette fonction. Elle définit un ensemble de solutions potentielles au problème.

On dit qu'une fonction f définie sur un ensemble A admet un **Minimum (resp. Maximum) global** au point M si pour tout point x de A on a : $f(M) \leq f(x)$ (resp. $f(M) \geq f(x)$). On dit que f admet un **Minimum (resp. Maximum) local** au point m s'il existe un voisinage V inclut dans l'ensemble A, et pour lequel pour tout point x satisfait : $f(m) \leq f(x)$ (resp. $f(m) \geq f(x)$).

Le **bassin d'attraction** d'un attracteur (minimum/maximum local/global) donné est l'ensemble des conditions initiales qui le rejoignent à mesure que le temps s'écoule.

L'ensemble des contraintes définit des conditions sur l'espace d'état que les variables doivent satisfaire. Ces contraintes sont souvent des inégalités ou des égalités isues directement du problème et permettant en général de limiter l'espace de recherche.

Une méthode d'optimisation recherche le point ou un ensemble de points de l'espace des états possibles qui satisfait au mieux un ou plusieurs critères traduit dans une ou plusieurs fonction objectifs. Le résultat est appelé *valeur optimale* ou *optimum*.

2.2 Les grandes méthodes d'optimisation

Comme nous l'avons vu brièvement au début de ce chapitre, il exite de nombreuses méthodes d'optimisation qui sont liées aux différents types de problèmes qui ont été rencontrés au cours du temps. Le choix d'une méthode est donc en liaison directe avec le problème à traiter et il n'en n'existe pas qui soit performante de manière universelle.

Devant l'étendue des méthodes, chaque école a sa façon de les classer. On peut par exemple noter les grands principes suivants :

- avec ou sans calcul de dérivée
- optimisation globale ou locale
- méthodes déterministes, énumératives, stochastiques (Goldberg [6])
- Neos guide optimization tree[2]

La classification retenue pour présenter les méthodes est basée sur la distinction entre *déterministe* et *stochastique*. Le tableau 2.1 donne brièvement l'idée générale de fonctionnement accompagnée d'exemples de méthodes ainsi que leurs grands avantages et inconvénients. Notons qu'une telle classification se veut générale et que la comparaison de méthode ne peut se faire que pour des problèmes donnés.

2.3 Le conflit exploration/exploitation

Les méthodes présentées dans le tableau 2.1 font intervenir chacune une part **d'exploration** et une part **d'exploitation** qui leur sont propre. Ces deux notions sont fondamentales en optimisation mais sont, hélas, contradictoires.

[2]Neos optimization tree : //http ://www-fp.mcs.anl.gov/otc/Guide/OptWeb/

Méthodes Déterministes	**Méthodes stochastiques**
Principe général	
⇒ *exploration méthodique de l'espace de recherche*	⇒ *processus de création aléatoire de points dans l'espace de recherche* ↪ une heuristique permet de guider la convergence
Exemples de méthodes	
Méthodes locales : → gradient (*state of the art*) → simplexe (Spendley *et al.* [20], Nelder & Mead [18])	→ méthodes Tabou (Glover [5])
Méthodes globales : → branch and bound → algorithme A^*	→ Monte Carlo → Recuit simulé → Algorithmes évolutionnaires (génétique : Goldberg [6], essaim d'abeilles, ...) → branch and bound stochastique
Avantages et inconvénients	
+ efficaces sur des problèmes de petites tailles + très utilisés pour leur robustesse + relativement aisés à implanter	+ flexibilité d'implantation + flexibilité par rapport aux contraintes + qualité élevée des solutions + parallélisable
- temps nécessaire pour trouver l'optimum global long ↪ bonne connaissance du problème pour fixer les variables d'initialisation	- nécessite beaucoup d'itérations - convergence vers un optimum en un temps fini non assurée

TAB. 2.1 – "Tentative de classification" des techniques d'optimisation

Par exploration, on entend l'aptitude de la méthode à parcourir avec efficacité l'espace d'état. Trop d'exploration augmente le temps de restitution de l'algorithme mais elle est nécessaire au maintient de la diversité et à la recherche d'un optimum global.

Derrière l'exploitation se cache l'aptitude d'une méthode à utiliser les résultats déjà obtenus pour faire converger l'algorithme.

Exploration	**Exploitation**
↪ méthode efficace ↪ permet la diversité dans les solutions	↪ méthode efficiente ↪ permet une convergence rapide vers un optimum
- résultat proche de l'optimum global - pas d'importance sur le temps de calcul - aucune partie de l'espace d'état non visitée	- résultat satisfaisant dans un temps court (local) - temps de calcul prime sur la précision - perte rapide de la diversité
→ Exemples : Méthodes Monte Carlo	→ Exemples : Gradient, Simplexe

TAB. 2.2 – L'exploration contre l'exploitation

Le choix d'une méthode d'optimisation doit donc passer par une réflexion sur ce conflit, tout en tenant compte des attentes du problème (TAB. 2.2) ainsi que des moyens de calcul dispo-

nibles. Notons tout de même que des méthodes de type "recuit simulé" ou encore "algorithme génétique" sont en quelque sorte une réponse à ce conflit. En effet, pour certains problèmes, il est possible d'ajuster leurs paramètres de fonctionnement (liés à l'heuristique) afin d'obtenir un bon compromis. Il est aussi tout à fait possible d'envisager des algorithmes hybrides qui "couplent" des techniques d'exploration avec des techniques d'exploitation. Cette vision amènera sans doute les meilleurs résultats sur les applications spécifiques sur lesquelles elles seront développées (travaux de L. Dumas [17]).

2.4 Le couplage CFD/optimisation

La montée en puissance des machines de calcul scientifique et leur accessibilité a permis de grandes avancées en Mécanique des Fluides Numérique. De plus en plus, les manipulations expérimentales sont remplacées par des calculs. Cette modélisation de la réalité donne naturellement un environnement confortable pour développer des applications d'optimisation qui se veulent à la base être programmables. Toutefois, le couplage de ces deux disciplines, CFD et optimisation, est un art qui demande beaucoup de ressources informatiques.

L'optimisation en Mécanique des Fluides concerne en grande partie le design des géométries (optimisation de forme des limites physiques imposées par la géométrie) mais aussi les conditions aux limites à appliquer sur le système (entrées et sorties). L'idée est de résoudre un problème inverse : un problème pour lequel on cherche les causes (typiquement, les conditions aux limites, la géométrie ...) connaissant les effets (objectifs que l'on se fixe). Pour un jeu de paramètres d'optimisation fixé, l'état dans l'espace de recherche (point courant) est obtenu par le biais d'un calcul CFD et de son interprétation en post-traitement qui donnera la valeur de la fonction objectif. Celle-ci n'est pas connue de manière analytique et par conséquent, l'accès à son gradient n'est pas direct. De plus, étant une fonction numérique, elle est souvent très bruitée. Les algorithmes d'optimisation ont besoin pour converger vers la solution optimale soit du gradient de la fonction objectif à ce point, soit de valeurs en différents points de l'espace de recherche que l'on va comparer avec le point courant.

La détermination du gradient de la fonction objectif en un point (pour des méthodes de type Newton, quasi-Newton...) peut se faire de diverses manières :

- la plus simple est l'utilisation des différences finies à l'ordre 2 qui implique le calcul d'autres états *proches* du point où l'on souhaite connaitre le gradient. Une des principale difficulté réside dans le concept de *proche* car l'incrément doit être d'ordre supérieur à la perturbation générée par les erreurs numériques mais assez petit pour signifier quelque chose
- pour palier aux problèmes de l'utilisation des différences finies, il est possible de procéder en variables complexes. Toutefois, cette méthode implique de reprogrammer le code en complexe
- une technique très performante et utilisée notament par B. Mohammadi [16] pour des travaux sur de l'optimisation de forme est la différenciation automatique. Le but est de créer à partir du code existant un code pour les dérivées. L'effort reside donc dans la génération de ce code mais son utilisation est ensuite très efficace

Les méthodes à gradient sont connues pour être les plus performantes dans les cas où elles s'appliquent. Toutefois, elles ne permettent pas de faire de l'optimisation multi-objectif, c'est à dire de concilier plusieurs objectifs à la fois (si ce n'est par la création d'une fonction objectif étant une combinaison des divers objectifs à atteindre). De plus, pour un grand nombre de paramètres d'optimisation, on se rend compte qu'elles deviennent coûteuses. Ces deux aspects, multi-objectif et grand nombre de paramètres, sont justement des caractéristiques de l'optimisation en CFD. De ce fait, les méthodes mettant en oeuvre des comparaisons d'état sont souvent utilisées (simplexe ou encore algorithme génétique).

Dans la littérature, il est possible de trouver des tests réalisés avec certaines des méthodes présentées dans le tableau 2.1. On peut notamment citer :

- un papier de Abramson *et al.* [1] qui présente une application utilisant des algorithmes de type BFGS (Broyden-Fletcher-Goldfarb-Shanno, ou quasi Newton), simplexe et recuit simulé pour de l'optimisation de forme sur une aile d'avion (fonction objectif basée sur la portance à maximiser et la traînée à minimiser).[3]
- un rapport de Hiroyasu et al. [8] qui montre l'utilisation d'un algorithme génétique sur l'optimisation multi-objectif d'un moteur diesel en vue d'une maîtrise de la pollution et de la consommation en fuel.
- les travaux de Bueche *et al.* sur la mise au point d'un algorithme génétique [11] et son utilisation dans le cadre d'optimisation multi-objectif (notion de diagramme de Pareto) à des processus de combustion [2] ainsi que du design de turbomachines [4].
- la thèse de S. Izquierdo [10] dont des résultats ont été présentés au congrès *International Congres on Evolutionary Methods for Design, Optimization and Control with Application to Industrial Problems EUROGEN* en 2003. Cette étude concerne la réduction de l'émission de polluant dans des chambres de combustion par l'utilisation d'un algorithme génétique dans le cadre du processus de post-combustion ainsi que pour un bruleur de type swirlé.
- un article de Ray & Tsai [19] portant sur le développement d'un algorithme évolutionnaire de type essaim d'abeille testé sur des cas d'optimisation de profils d'ailes.
- les travaux de Dumas [17] qui a opté pour une utilisation couplée de méthodes, tirant ainsi bénéfice du meilleur de chacune. La base de son algorithme est une méthode génétique sur laquelle il imbrique une méthode de descente pour guider l'opération de mutation cas et accélérer la convergence. Les tests effectués portent sur la réduction de traînée d'une voiture.

Les travaux présentés ci-dessus sont essentiellement basés sur des techniques évolutionnaires qui sont actuellement en plein essor. Bien que performantes si elles sont utilisées correctement (notion de probabilités pour certaines opérations), elles ne reposent sur aucun fondement mathématique. Leur principal intérêt est lié au fait que ce sont des méthodes de recherche globales, qui privilégient l'exploration mais sans laisser de coté l'exploitation. Elles permettent notamment la mise en évidence d'extrémums non habituels, et donc d'apporter des idées innovantes (alors que dans le cas d'un algorithme basé sur un gradient, on cherche juste à affiner une solution).

L'objectif que l'on s'est fixé pour ce stage était de constater les possibilités d'intégration du code N3SNatur dans une boucle d'optimisation automatique. Le code CFD doit alors être vu comme une boite permettant de calculer la fonction objectif en un point de l'espace d'état, quelque soit l'algorithme d'optimisation utilisé (FIG. 2.1).

TURBOMECA ayant des ambitions liées à l'amélioration des solutions existantes et non pas à l'exploration de multiples possibilités innovantes, l'algorithme choisit pour tester l'implantation du code N3SNatur dans une boucle d'optimisation est l'algorithme du simplexe. Il a l'avantage principal d'être simple à mettre en oeuvre ce qui permettra donc une validation rapide de la méthode utilisée dans le cadre du stage pour le couplage entre CFD et optimisation (voir Chapitre 3).

[3] deux liens internet sur des logiciels d'optimisation :
DAKOTA http ://endo.sandia.gov/DAKOTA/index.html
NEOS http ://www-neos.mcs.anl.gov/neos/

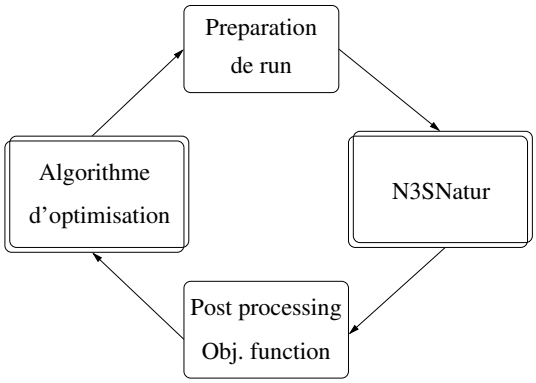


FIG. 2.1 – Implantation de N3SNatur dans un algorithme d'optimisation

2.5 L'algorithme du Simplexe : premier choix retenu

Dans cette section, un bref historique sur l'origine de l'algorithme du simplexe précède l'exposé de celui utilisé pour la mise oeuvre de la boucle d'optimisation.

2.5.1 Origine de la méthode du Simplexe

Direct Search :

Hooke & Jeeves [9] décrivent les méthodes de recherche directe dont fait partie celle du simplexe de la façon suivante :

direct search describes sequential examination of trial solutions involving comparison of each trial solution with the "best" obtained up to that time together with a strategy for determining (as a function of earlier results) what the next trial solution will be. The phrase implies our preference, based on experience, for straightforward search strategies which employ no techniques of classical analysis except where there is a demonstrable advantage in doing so.

La recherche en optimisation a fait émerger au cours du temps des solutions efficaces pour un certain nombre de problèmes. Cependant, on constate que les méthodes de recherche directe sont toujours très utilisées à l'heure actuelle. Ceci s'explique par les principaux phénomènes suivants :

- elles donnent de bons résultats en pratique
- elles sont applicables à des problèmes non linéaires et fonctionnent bien là ou des techniques plus sophistiquées échouent
- souvent, elles sont utilisées en premier recours car elles sont simples d'implantation et d'utilisation
- elles peuvent permettre de "dégrossir" un problème et de donner une condition initiale pour des méthodes plus complexes

Ces méthodes sont basées sur quelques concepts qui expliquent leur simplicité et leur efficacité : elles ne demandent pas d'évaluation de dérivée (*zero-order method*) mais fonctionnent sur les valeurs de la fonction coût. Généralement, c'est simplement l'ordre relatif des points (classement par rapport à la valeur de la fonction objectif en ces points) dans l'espace de recherche qui est important. Dans la recherche d'extrémum, n'importe quel point meilleur que les autres est accepté sans condition. Pour résumer, elles sont lentes à converger, ne possèdent pas de théorème de convergence mais sont robustes et très utilisées.

Simplex search :

La première version du simplexe est due à Spendley *et al.* [20] et est motivée par une réduction du coût (nombre d'évaluations de la fonction objectif) par itéré. n étant la dimension de l'espace de recherche, les méthodes de recherche directe antérieures nécessitaient de $2n$ à 2^n évaluations par itéré alors qu'il est possible de n'en faire que $n+1$. Ce nombre d'évaluations correspond :

- au nombre de points nécessaires pour faire un plan dans un espace à n dimensions
- au nombre de points utiles à l'estimation de la dérivée première de la fonction objectif par différence finies
- au nombre de sommets d'un simplexe dans un espace de dimension n

L'idée est donc de construire un simplexe non dégénéré dans l'espace de recherche et de s'en servir pour mener la recherche. Un simplexe est une figure géométrique qui possède $n+1$ sommets dans un espace de recherche de dimension n et qui doit pouvoir être le support d'une base de cet espace pour être considéré comme non dégénéré.

La version originale de Spendley *et al.* [20] ne contient qu'une seule opération : les $n+1$ sommets du simplexe étant classés selon la valeur de la fonction objectif qui leur est associée, on cherche à remplacer le moins bon ($n+1^{ième}$) à chaque itération. Pour cela, il est réfléchi par rapport au centroide des autres. Si le résultat correspond à un sommet toujours plus mauvais que les n autres, alors cette opération dite de réflexion (FIG. 2.2) doit être refaite avec le sommet classé en avant dernière position et ainsi de suite. On se rend vite compte avec quelques essais qu'appliquer la méthode telle que décrite ci-dessus amène rapidement le simplexe à s'enfermer dans un cycle infini. Il existe certaines parades qui ont été développées pour améliorer ce point mais les plus efficaces sont certainement l'ajout d'opérations.

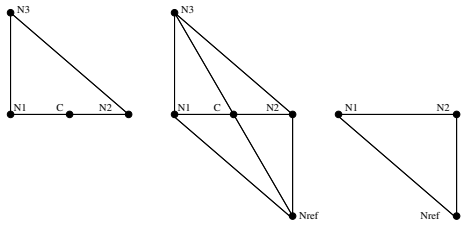


FIG. 2.2 – Opération de réflexion dans l'algoritme de Hooke & Jeeves

Nelder & Mead [18] ont en effet proposé une amélioration de l'algorithme du simplexe en augmentant le nombre d'opérations possibles en vue d'accélérer la convergence. La réflexion était une bonne base car elle conserve la géométrie initiale du simplexe qui ne dégénère alors pas. Toutefois, la recherche ne s'adapte pas à la forme de la fonction. Il est donc nécessaire de changer la forme du simplexe par l'ajout d'opérations. La figure 2.3 montre ces opérations qui sont : l'expansion, la contraction interne et externe, et le rétrécissement du simplexe. (Notons bien qu'à chaque opération correspond un run N3SNatur sauf dans le cas de celle de rétrécissement qui en compte n).

Les opérations de contraction et d'expansion adaptent l'orientation et la taille du simplexe à la recherche et par conséquent, rendent mieux en compte du gradient de la fonction objectif. Le rétrécissement a lieu lorsque les autres options n'aboutissent pas à de meilleurs sommets : l'échelle de recherche est alors réduite par la diminution de la taille du simplexe.

L'algorithme de Nelder & Mead (détaillé dans la section 2.5.2) est un séquencement des opérations réalisé de façon astucieuse. Il est très performant sur certains problèmes permettant moins d'évaluations de la fonction objectif que d'autres méthodes. Toutefois, il arrive que la

méthode ne fonctionne pas : la littérature parle souvent de la lenteur de sa convergence, ou de convergence prématurée. Une analyse fine sur des fonctions tests montre que les déformations du simplexe peuvent amener à obtenir une direction de recherche (*i.e.* direction définie par le moins bon vertex et le centroide des autres sommets) orthogonale au gradient de la fonction. Ceci est notamment le cas lors de l'étude de fonctions qui présentent une faible pente (fonction souvent utilisée pour mettre à défaut l'algorithme : fonction de Rosenbrock). De nombreux travaux ayant été effectués sur cette méthode, on a pu montrer sa robustesse et sa convergence lorsque l'on travaille à l'ordre $n = 1$. Quelques propriétés ont pu être établies à des ordres plus élevés mais il n'existe pas de théorème de convergence globale. Mc. Kinnon [15] a même montré qu'il est impossible de prouver la convergence globale de cet algorithme en grande dimension.

Une des grandes équipes qui travaille actuellement sur le développement des méthodes de type simplexe est celle de Torczon[4]. L'algorithme utilisé et présenté ci-dessous a été inspiré de leurs travaux [7].

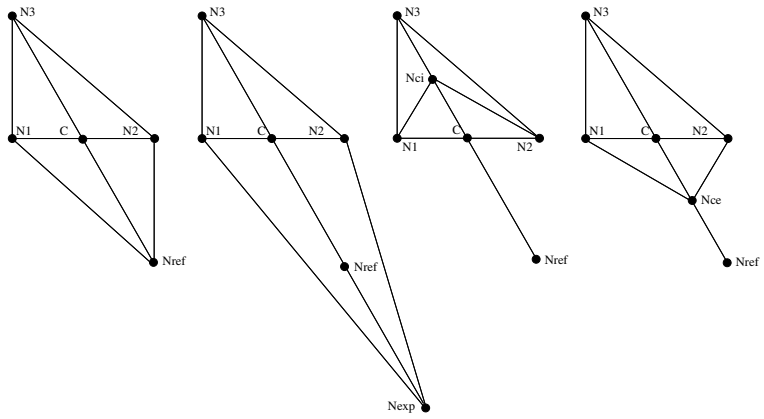


FIG. 2.3 – Opérations de l'algorithme de Nelder & Mead : réflexion Nref, expansion Nexp, contraction interne Nci, contraction externe Nce

2.5.2 Algorithme utilisé : Nelder & Mead

Initialisation du simplexe : n+1 vertex N_i et Calcul de la fonction objectif $f_{obj}(N_i)$ pour chacun d'eux

Classement du simplexe : $f_{obj}(N_1) \leq f_{obj}(N_2) \leq ... \leq f_{obj}(N_n) \leq f_{obj}(N_{n+1})$

Tant que non convergence :

$\Rightarrow$ Détermination du centroide : $C^* = \frac{1}{n}\sum_{i=1}^{n} N_i$

$\Rightarrow$ *Opération de réflexion* : $N_{ref} = 2C^* - N_{n+1}$ et calcul de $f_{obj}(N_{ref})$

$\hookrightarrow$ si $(f_{obj}(N_1) \leq f_{obj}(N_{ref}) \leq f_{obj}(N_n))$ alors la réflexion est acceptée : $N_{n+1} = N_{ref}$

$\hookrightarrow$ aller à *Classement du simplexe*

$\Rightarrow$ *Opération d'expansion* : si $(f_{obj}(N_{ref}) < f_{obj}(N_1))$

$\hookrightarrow$ $N_{exp} = C^* + 2(N_{ref} - C^*)$ et calcul de $f_{obj}(N_{exp})$

$\hookrightarrow$ si $(f_{obj}(N_{exp}) < f_{obj}(N_{ref}))$ alors expansion acceptée : $N_{n+1} = N_{exp}$ sinon réflexion

[4]College William & Mary, Williamsburg. Computer and Science Team

acceptée : $N_{n+1} = N_{ref}$

$\hookrightarrow$ aller à *Classement du simplexe*

$\Rightarrow$ *Opération de contraction externe* : si $(f_{obj}(N_n) \leq f_{obj}(N_{ref}) < f_{obj}(N_{n+1}))$

$\hookrightarrow$ $N_{ce} = C^* + \frac{1}{2}(N_{ref} - C^*)$ et calcul de $f_{obj}(N_{ce})$

$\hookrightarrow$ si $(f_{obj}(N_{ce}) \leq f_{obj}(N_{ref}))$ alors contraction externe acceptée : $N_{n+1} = N_{ce}$. Aller à *Classement du simplexe*

$\hookrightarrow$ sinon : aller à *Opération de rétrécissement*

$\Rightarrow$ *Opération de contraction interne* : si $(f_{obj}(N_{ref}) \geq f_{obj}(N_{n+1}))$

$\hookrightarrow$ $N_{ci} = C^* + \frac{1}{2}(N_{n+1} - C^*)$ et calcul de $f_{obj}(N_{ci})$

$\hookrightarrow$ si $(f_{obj}(N_{ci}) < f_{obj}(N_{n+1}))$ alors contraction interne acceptée : $N_{n+1} = N_{ci}$. Aller à *Classement du simplexe*

$\hookrightarrow$ sinon : aller à *Opération de rétrécissement*

$\Rightarrow$ Si (*Opération de rétrécissement*) demandée :

$\hookrightarrow$ pour i de 1 à n, remplacer N_i par $N_i + \frac{1}{2}(N_1 - N_i)$

$\hookrightarrow$ calculer $f_{obj}(N_i)$ pour les nouveaux N_i

$\Rightarrow$ *Classement du simplexe* : $f_{obj}(N_1) \leq f_{obj}(N_2) \leq ... \leq f_{obj}(N_n) \leq f_{obj}(N_{n+1})$

$\Rightarrow$ *Test de convergence* : l'arrêt de la méthode est lié d'une part à un nombre maximum d'itérations admissible ainsi qu'à un critère de convergence. On aurait pu imaginer stopper l'algorithme sur un critère lié à la taille du simplexe : en effet, à convergence les sommets du simplexe tendent vers un même point de l'espace de recherche. Toutefois, le critère choisit repose sur la fonction objectif. On introduit μ, la valeur moyenne des fonctions objectifs correspondant aux sommets du simplexe excepté le meilleur : $\mu = \frac{1}{n}\sum_{i=2}^{n+1} f(N_i)$ et ϵ une constante de tolérance. La méthode prend fin lorsque :

$$\left[\frac{1}{n+1}\sum_{i=1}^{n+1}[f(N_i) - \mu]\right]^{\frac{1}{2}} < \epsilon \tag{2.1}$$

Fin de tant que

Afin de restreindre l'espace de recherche, les variables d'optimisations possèdent des bornes supérieures et inférieures. Si une des opérations de la méthode aboutit à un point situé à l'extérieur de l'espace d'état, alors il se verra attribuer une valeur de fonction objectif très élevée. Le simplexe sera ainsi forcé à ne pas continuer sa recherche hors de cet espace. Les bords de domaines ainsi pris en compte peuvent être vu par l'algorithme comme des minimums locaux.

Des tests permettant de valider l'écriture du code et d'évaluer le domaine d'application de l'algorithme ont été réalisé et sont présentés en annexe A.

Chapitre 3

Mise en place de l'outil d'optimisation

Le projet d'insertion du code N3SNatur dans une boucle d'optimisation automatique est présenté comme étant une tâche de couplage d'un code de CFD avec un code d'optimisation. En effet, il a été vu au chapitre 2 que le code N3SNatur est entièrement imbriqué dans l'algorithme d'optimisation. C'est pour cette raison qu'un coupleur de code, PALM, a été utilisé pour réaliser l'outil.

3.1 Le code de CFD : N3SNatur

Le code N3SNatur résout les équations de Navier-Stokes multi-espèces avec prise en compte de la turbulence suivant un modèle du premier ordre à deux équations (modèle $k - \epsilon$, avec k l'énergie cinétique turbulente et ϵ son taux de dissipation). La résolution peut se faire soit sur un domaine fixe, soit sur un domaine mobile. Le second cas permet de prendre en compte les géométries déformables.

Les modélisations physiques permettent la simulation d'écoulement :
- Compressible avec des Mach compris entre 0.01 et 5,
- Turbulent ou laminaire multi-espèces,
- Reactifs (certains modèles de combustion sont disponibles),
- De fluides réels, calorifiquement ou thermiquement parfaits

Les méthodes de discrétisation sont basées sur des triangles ou des thétraèdres (version multi-élément en cours de développement en collaboration avec le CERFACS) mixant des éléments finis et des volumes finis. Les méthodes d'intégrations temporelles et spaciales (MUSCL) sont d'ordre 2 ou 3, explicites ou implicites (choix utilisateur).

N3SNatur est un code parallèle qui utilise pour ses communications soit du PVM soit du MPI et peut être lancé sur des machines de calcul parallèle. N3SNatur est écrit en Fortran (77 et 90).

Pour plus de détails concernant les équations codées dans N3SNatur ou sur son utilisation, le lecteur peut se reporter au manuel théorique [13] et au manuel utilisateur [14].

3.2 Le coupleur : PALM

PALM (Projet d'Assimilation par Logiciel Multi-méthode) est un coupleur dynamique de codes parallèles développé par l'équipe *Global Change* du CERFACS. Son origine est liée au projet d'océanographie opérationnel MERCATOR qui vise à la mise en oeuvre d'une chaîne

d'assimilation de données qui soit flexible et performante.

Le formalisme de PALM est de créer des unités indépendantes qui consomment et produisent des données via des primitives telles que *PALM_Get* et *PALM_Put*. Une même unité, qui représente soit une tâche élémentaire soit un programme à coupler, peut être intégrée dans des applications différentes sans aucune modification. Pour construire les applications, les unités sont séquencées dans des branches. Celles-ci peuvent contenir des structures de contrôle leur permettant ainsi de répéter en boucle une partie de la séquence ou de faire des exécutions conditionnelles. Un schéma de communication décrivant l'échange de données entre unité doit alors être créé. La constitution de l'algorithme de couplage est réalisable à partir de l'interface graphique PrePALM (FIG. 3.2).

L'indépendance des unités garanti la modularité qui est une caractéristique essentielle car comme on l'a vu au chapitre 2, l'algorithme d'optimisation utilisé est amené à changer. De plus, le couplage est dit dynamique car PALM peut lancer des composants en cours de simulation. PALM permet aussi de changer la distribution parallèle des composants. La flexibilité du coupleur est donc un atout majeur pour son utilisation. Concernant les performances, PALM possède deux niveaux de parallélisme (FIG. 3.1) :

- un premier au niveau des branches et qui représente une séquence d'instruction (unités, programmes shell). Typiquement, deux actions différentes peuvent être exécutées en parallèle sur deux branches différentes
- un second au niveau des unités qui peuvent être distribuées

Ces deux niveaux sont primordiaux car on peut ainsi imaginer lancer sur plusieurs branches des calculs N3SNatur eux même parallèles. Enfin, PALM repose sur des bibliothèques de communication parallèle MPI offrant des possibilités de communications à hautes performances.

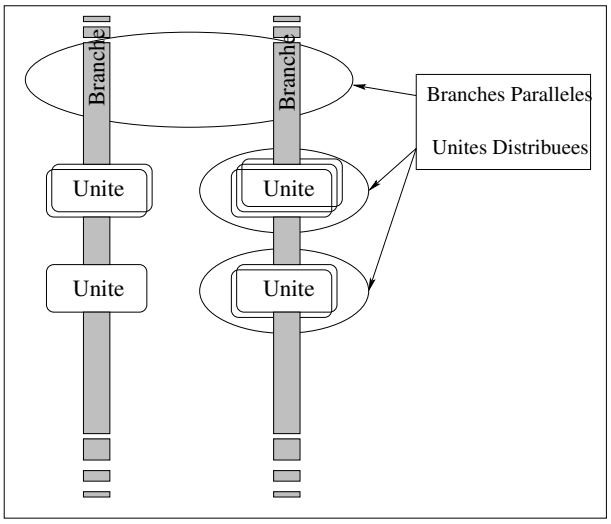


FIG. 3.1 – Les deux niveaux de parallélisme de PALM

Quelques coupleurs de code plus ou moins analogues à PALM sont en cours de développement dans des buts de couplage multi-physique de code (couplage fluide-structure). L'école d'été EDF-CEA-INRIA, qui s'est déroulée en juin 2004 à Saint Lambert des Bois (78), portait sur ce thème et permettait une présentation de certains de ces coupleurs : Salomé (EDF-CEA), Calcium (EDF), PCCI (SCAI).

PALM a été exposé comme un outil très reconnu et efficace en avance sur ses concurrents au niveau des techniques de couplage. Le choix du coupleur est donc, dans cette phase de conception d'outil, optimal.

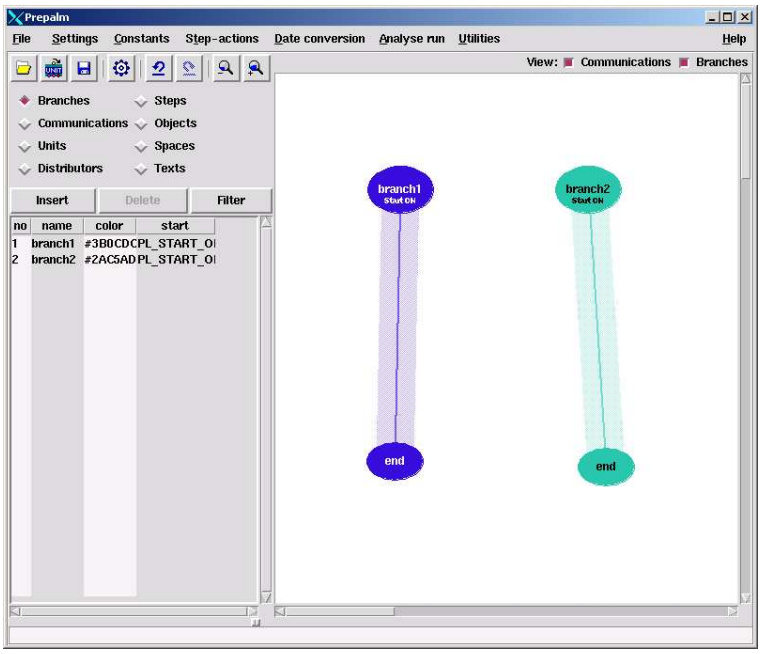


FIG. 3.2 – Interface graphique PrePALM

3.3 Intégration de N3SNatur dans PALM

La première étape pour utiliser un code dans PALM est de le rendre compatible avec le coupleur. Cette tâche fut donc plutôt de type informatique et avait pour but de créer une unité N3SNatur. Ensuite, l'algorithme du simplexe a été implanté sous PALM intégrant l'unité N3SNatur pour le calcul de la fonction objectif.

3.3.1 Partie informatique

Le parallélisme MPI du code N3SNatur

Pour comprendre les actions à effectuer pour créer une unité PALM à partir du code N3SNatur, il faut entrer dans le parallélisme du code. La communication entre les processus est basée sur les librairies MPI (Message Passing Interface). Ces librairies permettent notament l'activation de tous les processes en début d'exécution et leur libération à la fin.

Dans N3SNatur, un process est considéré comme le maître : il donne du travail aux autres processes (esclaves) et attend un signal de fin pour achever le run. Les esclaves font les mêmes opérations mais sur des parties de domaines (le maillage est divisé en autant de parties qu'il y a d'esclaves).

Les communications entre le maître et les esclaves (instruction de fin de run) ou entre les esclaves (conditions de bord sur les sous-domaines) se fait par le biais de MPI et de ce que l'on appelle les groupe et communicateurs.

Notion de groupe et de communicateur MPI

Une spécificité de MPI est son utilisation de communicateurs, qui peuvent se rapprocher de la notion d'objets appliqués à la programmation MPI. Un communicateur est un objet opaque, il contient des informations cachées à l'utilisateur mais qu'il peut interroger par l'intermédiaire

de primitives. Chaque communicateur MPI regroupe un ensemble de processus que l'on a décidé d'unir dans le but de simplifier les échanges d'informations (mais aussi les synchronisations ou autre actions liées au parallélisme). Les processus possèdent un rang unique dans les communicateurs qui leur sert d'identifiant (un même processus peut appartenir à plusieur communicateur avec des identifiantq différents). Par exemple, l'objet MPI_COMM_WORLD, qui est un communicateur, contient tous les processus qui ont démarré dans une machine virtuelle donnée. Ces derniers peuvent lui demander combien de processus ont démarré et par exemple quel est leur identifiant.

Les intérêt des communicateurs sont multiples :

- chaque communication MPI a lieu dans le cadre d'un communicateur
- ils définissent alors un espace de communication sûr et fiable car chaque communicateur est indépendant des autres et les communications ne peuvent pas s'interférer d'un communicateur à l'autre
- ils définissent la portée d'une communication : ils contiennent des informations sur le contexte de communication (informations cachées à l'utilisateur)

Dans N3SNatur, les groupes et les communicateurs sont créés de la façon suivante :

- MPI_COMM_WORLD : le communicateur contenant tous les processes
- MPI_GROUP_WORLD : le groupe créé à partir du communicateur MPI_COMM_WORLD par la primitive MPI_COMM_GROUP
- NATUR_GROUP_SLAVE : le groupe d'esclaves créé à partir du groupe MPI_GROUP_WORLD et du nombre de processeurs à lui allouer (nombre de sous domaines demandés) par la primitive MPI_GROUP_INCL
- NATUR_COMM_SLAVE : le communicateur des esclaves créé à partir du groupe NATUR_GROUP_SLAVE et du communicateur MPI_COMM_WORLD par la primitive MPI_COMM_CREATE

Tous les communicateurs sont mis en place à partir de MPI_COMM_WORLD qui a donc un rôle important dans le code.

Les changements apportés aux source du code N3SNatur

Le code N3SNatur doit être vu par PALM comme une subroutine. Il est donc nécessaire en premier lieu de changer l'appellation *PROGRAM* en *SUBROUTINE*. Ensuite, le parallélisme est géré par PALM : toutes les procédures de création (*MPI_INIT*) et de destruction (*MPI_ABORT, MPI_FINALIZE*) de processus sont à supprimer ou à remplacer par des instructions codées dans les librairies de PALM (*PALM_ABORT*). Enfin, dans un contexte purement théorique, la dernière manipulation de code est de remplacer le communicateur MPI_COMM_WORLD (qui à présent référencie la totalité des processeurs et pas uniquement ceux de N3SNatur) par un communicateur prédéfini par PALM pour l'insertion d'unités parallèles : PL_COMM_EXEC.

Etant donné le peu d'occurence des appels aux procédures d'initialisation et de destruction de processus, les changements les concernant ont été réalisés directement en éditant le code source. Le passage de *SUBROUTINE* à *PROGRAM* a été également réalisé de cette façon. Par contre, un script shell permet le remplacement du communicateur MPI_COMM_WORLD par PL_COMM_EXEC dans toutes les routines concernées.

Afin de ne conserver qu'une seule version des sources du code, les modifications effectuées sont conditionnées par une clef de compilation. C'est en effet lors de la compilation que l'on choisit ou non de coupler le code avec PALM. Dans un script écrit durant ce stage et permettant de compiler toutes les sources du code (le noyau écrit en Fortran 77 et celui écrit en 90), une variable d'environnement (*COMP_CHOICE*) permet de faire le choix de compilation avec PALM. Si $COMP\_CHOICE = PALM$ alors :

- le script de changement de MPI_COMM_WORLD en PL_COMM_EXEC dans les sources est lancé
- la clef de compilation -DWITHPALM est activée (et donc toutes les modifications présentées précedemment sont prises en compte)
- les librairies et les "includes" de PALM sont linkés

Une librairie n3s_natur.a est constituée à partir de ces sources compilées et servira dans la création de l'exécutable de couplage.

Au delà de ces changements que l'on peut nommer "accadémiques", des modifications liées à la programmation de N3SNatur ont été nécessaires. Ils sont développés en annexe B et concernent principalement : la désallocation de tableaux et la restitution de tous les processeurs à PALM en fin d'exécution du code N3SNatur.

3.3.2 Partie algorithmique

Une fois l'unité N3SNatur crée et son fonctionnement sous PALM validé, elle a été incorporée à l'algorithme du simplexe prélablement implanté et testé dans PALM. La figure 3.3 représente le canevas PrePALM du couplage entre l'optimisation et le code de CFD et la figure 3.4 décrit le fonctionnement des deux branches qui composent ce couplage. Notons que ce canevas est une première version qui doit être améliorée : le but de l'utilisation de PALM est que chacun des programmes utilisé soit intégré en tant qu'unité PALM. Ces unités peuvent échanger des informations entre elles ou avec les branches par l'intermédiaire de fonctionnalités PALM prévues à cet effet. Dans le cas présenté, certains programmes de pre-postraitement de N3SNatur sont directement intégrés en tant qu'exécutables (facilite le transfert d'informations entre programmes qui se fait alors par fichiers, sources non disponibles ...) qui sont lancés via des scripts shell dans le code de branche. L'exécution de la séquence s'en retrouve ralentie.

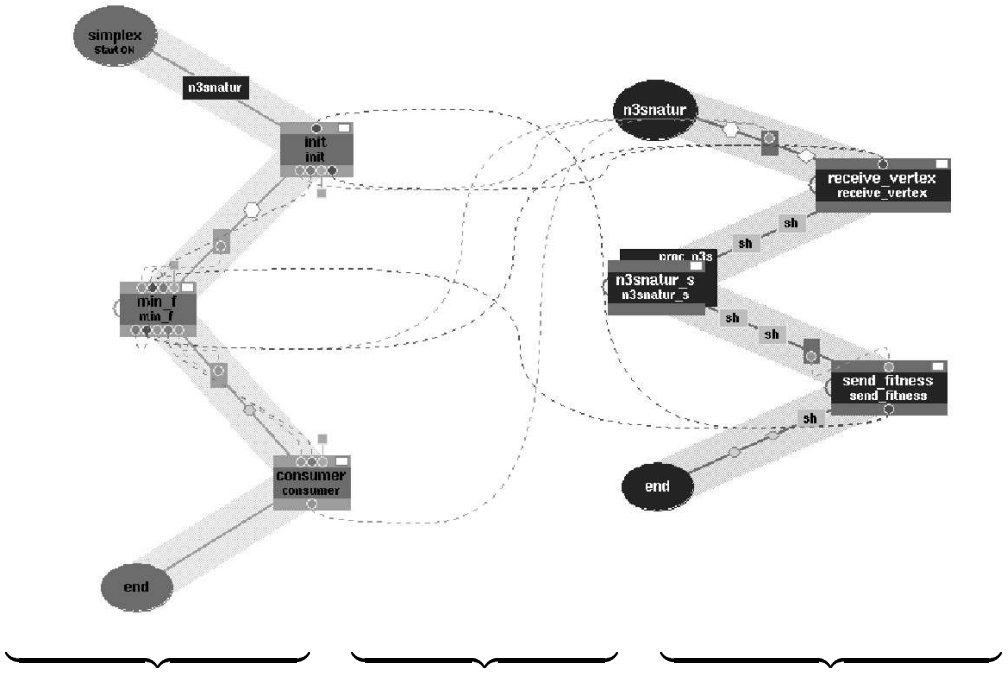


FIG. 3.3 – Canevas de la boucle de optimisation/N3SNatur sous PrePALM

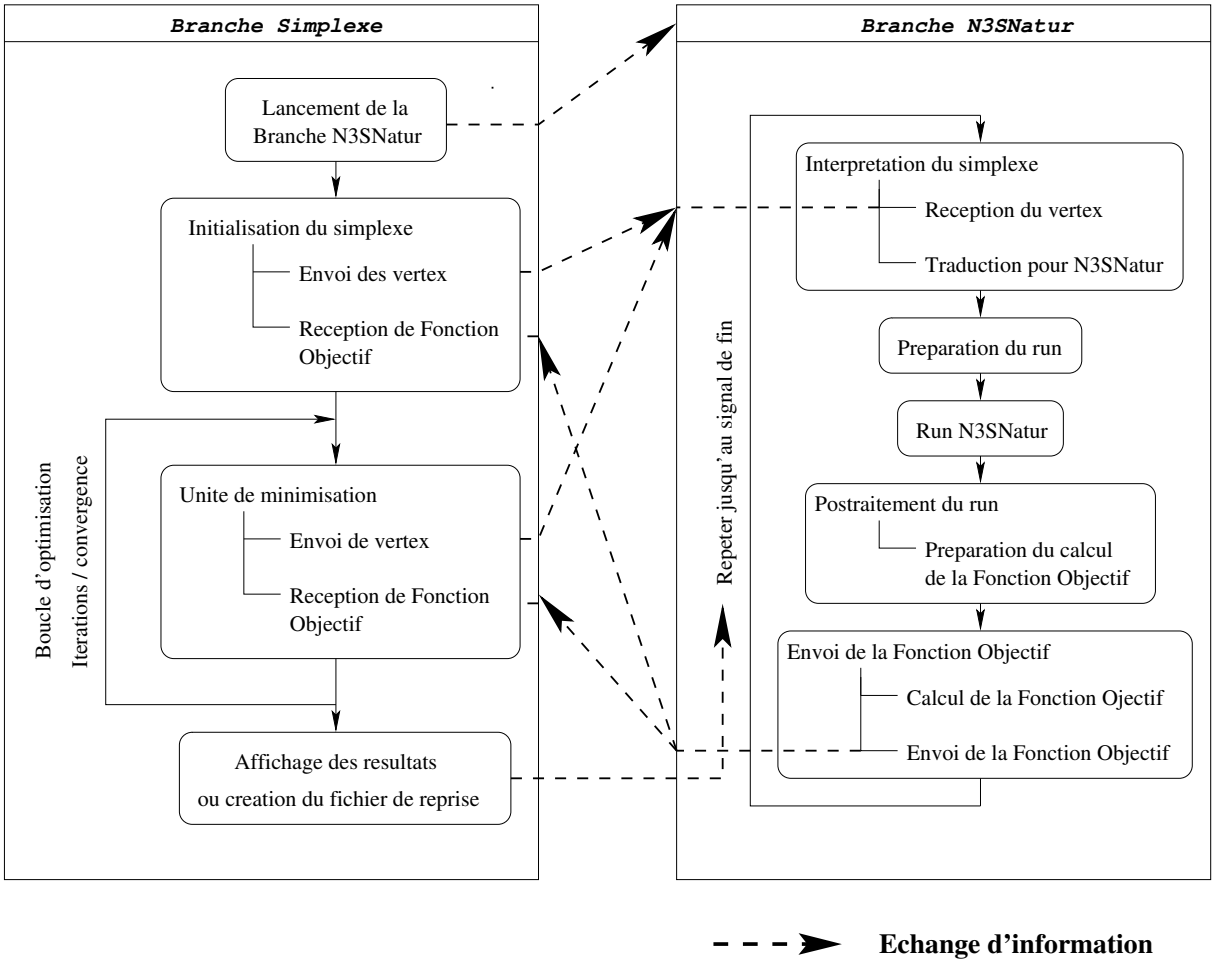


FIG. 3.4 – Étapes de l'algorithme de couplage

Chapitre 4

Runs d'optimisation

En accord avec TURBOMECA, une configuration simple dans l'esprit de celles qui feront l'object des futures études a été mise en place dans le but de tester le couplage entre l'algorithme d'optimisation et celui de CFD par PALM. Ce chapitre présente ce cas test ainsi que certains des runs d'optimisations qui ont été effectués.

4.1 Configuration étudiée

4.1.1 Géométrie et maillage

La configuration est basée sur un canal à deux dimensions dans lequel entre du gaz chaud que l'on souhaite refroidir par l'injection de gaz frais (FIG. 4.1). Le maillage utilisé, non structuré, comporte 5730 noeuds pour 11016 éléments ($0.5 \times 0.06\ m$).

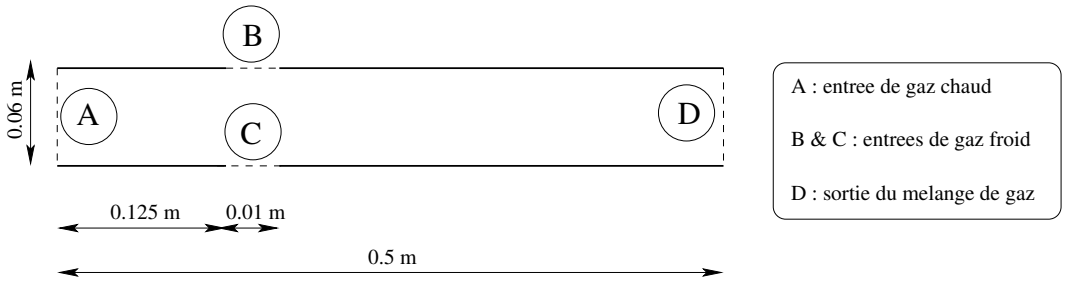


FIG. 4.1 – Géométrie des cas tests

4.1.2 Boucle d'optimisation

La figure 4.2 présente la méthode adoptée pour les calculs d'optimisation. Les variables d'optimisation potentielles sont : V_B, V_C, T_B et T_C. Pour chaque cas étudié, le profil de sortie cible qui permet de déterminer la valeur de la fonction objectif est issu du résultat d'un run N3SNatur. La fonction objectif est prise comme étant :

$$F_{obj} = \alpha \left(1 - \frac{T_{Calcul}}{T_{cible}}\right)^2 \tag{4.1}$$

A la différence d'un *RMS* standard, l'adimensionnalisation de l'écart entre la cible et le calcul permet l'attribution d'une importance relative à ces écarts. Des tests ont permis de montrer son bon comportement.

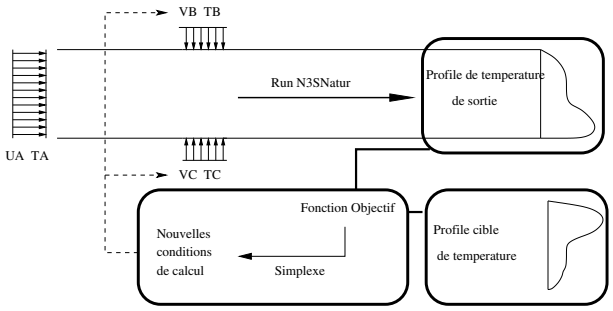


FIG. 4.2 – Description de la boucle d'optimisation

4.1.3 Calculs préliminaires

Des calculs préliminaires permettant d'optimiser le nombre d'itérations et donc le temps de restitution des runs N3SNatur (partie la plus coûteuse de l'optimisation) sont présentés en annexe D, accompagnés de calculs analytiques visant à déterminer la température en sortie de l'installation. Les paramètres physiques et numériques utilisés pour ces simulations y sont également détaillés.

4.1.4 Description des runs présentés

Les runs ont été menés par ordre croissant de difficulté (forme de la fonction objectif, nombre de paramètres). Ils correspondent aux souhaits soumis par TURBOMECA pour tester les capacités de l'outil d'optimisation. Pour chacun d'entre eux, on donne les paramètres d'optimisation (indiqués par * en optimisation 1D ou ^{kieme} correspondant à numérotation sur les figures de résultat), l'espace d'état et les valeurs utilisées pour obtenir la cible.

Les simplexes sont initialisés de manière à être orthogonaux (voir annexe A) et sont placés dans les bassins d'attraction des minimums recherchés.

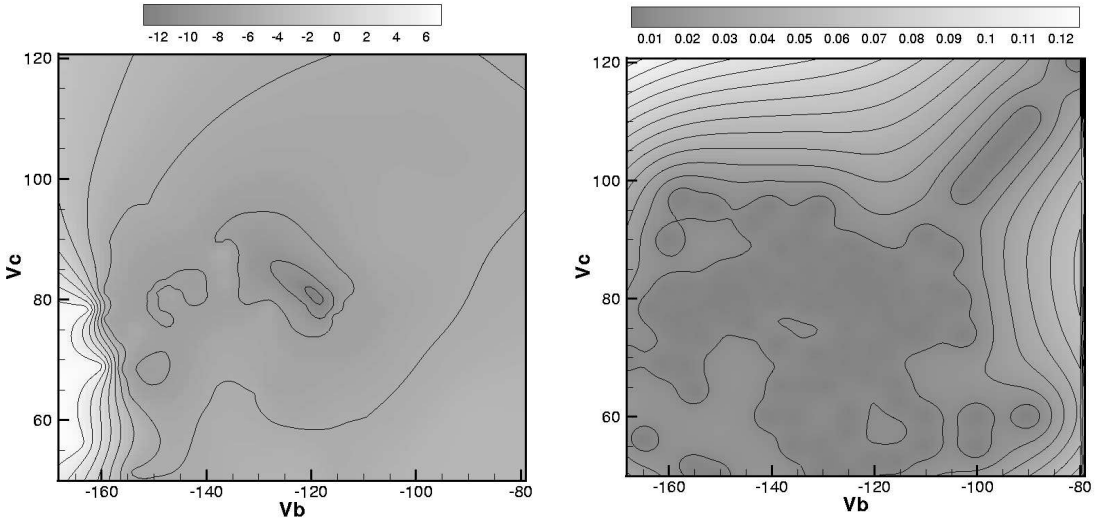
4.1.5 Présentation et lecture des résultats

Pour chaque run d'optimisation, deux séries de figures seront présentées :
- la première série représente l'évolution du simplexe, vertex par vertex, et la valeur de la fonction objectif en fonction des itérations
- la seconde est composée de deux figures : la première représente le meilleur profil à chaque itération de l'algorithme. La deuxième retrace le profil donné par les itérations. En effet, on a vu au chapitre 2 qu'à chaque itération, le moins bon des vertex est remplacé par un meilleur. Celui trouvé en fin d'itération n'est alors pas forcément le meilleur de tous les vertex.

4.2 Utilisation du krigeage

Le krigeage est une méthode optimale d'interpolation spatiale (voir annexe C) qui dans le cadre des applications présentées ici peut permettre de comprendre le comportement du simplexe. En effet, après plusieurs tentatives de run d'optimisation sur le cas 2D "Profil de Température asymétrique" qui ne convergeaient pas vers le minimum attendu, un programme de krigeage à été utilisé. Il en est ressortit l'existance de plusieurs minimums locaux et un minimum global sur l'espace d'état utilisé. La figure 4.2 montre le résultat du programme de krigeage appliqué aux vertex trouvés après plusieurs runs d'optimisation.

A l'aide de ce résultat, il est possible de placer le simplexe initial dans le bassin d'attraction du minimum qui est recherché. C'est de cette façon qu'ont été initialiés les calculs 1D "Débit d'injection total donné", 2D "Profil de Température asymétrique" et dans une moindre mesure 3D "Profil de Température asymétrique".



(a) (b)

FIG. 4.3 – Résultat du programme de krigeage : fonction objectif pour une cible $V_B = -120$ ms^{-1} et $V_C = 80$ ms^{-1} (a) et variance de l'erreur associée (b)

4.3 Optimisation à un paramètre

Débit en B et C égaux (FIG. 4.4 et 4.5 page 25)

Ce test est très simple et permet de valider le bon fonctionnement du couplage. Les paramètres de ce run sont :

V_B	T_B	V_C^*	T_C	**Cible**
$-V_C$	300 K	$\in [40, 150]$ ms^{-1}	300 K	$V_C = 100$ ms^{-1}

Débit en B et C égaux avec une cible impossible à atteindre (FIG. 4.6 et 4.7 page 26)

Pour cet essai, le but était de comprendre comment l'algorithme se comporte lorsque le but à atteindre est impossible.

V_B	T_B	V_C^*	T_C	**Cible**
$-V_C$	300 K	$\in [40, 150]$ ms^{-1}	300 K	$V_C = 80$ ms^{-1} et $V_B = -120$ ms^{-1}

Débit d'injection total donné (FIG. 4.8 et 4.9 page 27)

Il sagit d'un cas qui se rapproche d'une configuration réelle ou l'on se fixe une consommation en gaz frais.

V_B	T_B	V_C^*	T_C	**Cible**
$V_B + V_C = 200$ ms^{-1}	300 K	$\in [40, 150]$ ms^{-1}	300 K	$V_C = 80$ ms^{-1} et $V_B = -120$ ms^{-1}

4.4 Optimisation à deux paramètres

Profil de température symétrique (FIG. 4.10 et 4.11 page 28)

Pour commencer avec l'optimisation 2D, ce cas simple est testé :

V_B^{2ieme}	T_B	V_C^{1ier}	T_C	**Cible**
$\in [-150, -40]\ ms^{-1}$	$300\ K$	$\in [40, 150]\ ms^{-1}$	$300\ K$	$V_C = 100\ ms^{-1}$ et $V_B = -100\ ms^{-1}$

Profil de température asymétrique (FIG. 4.12 et 4.13 page 29)

Ce test fait intervenir plusieurs minimums locaux. La convergence vers le global est conditionnée par une initialisation du simplexe dans son bassin d'attraction (utilisation du krigeage).

V_B^{2ieme}	T_B	V_C^{1ier}	T_C	**Cible**
$\in [-150, -40]\ ms^{-1}$	$300\ K$	$\in [40, 150]\ ms^{-1}$	$300\ K$	$V_C = 80\ ms^{-1}$ et $V_B = -120\ ms^{-1}$

4.5 Optimisation à trois paramètres

L'optimisation 3D fait apparaître certaines limites de l'algorithme du simplexe. La technique de krigeage a été utilisée pour initialiser le simplexe dans le bassin d'atrraction du minimum global du cas test "Profil de température asymétrique".

(FIG. 4.6 et 4.6 page 30)

V_B^{2ieme}	T_B	V_C^{1ier}	T_C^{3ieme}	**Cible**
$\in [-150, -40]\ ms^{-1}$	T_C	$\in [40, 150]\ ms^{-1}$	$\in [270, 300]\ K$	$V_C = 80\ ms^{-1}\ V_B = -120\ ms^{-1}$ $T_C = 300\ K$

4.6 Résumé des runs

Tous les runs présentés ont été effectués sur 7 processeurs d'une machine SGI Origin 2000 (1 pour le driver de PALM qui gère les communications entre les unités, 1 pour la branche du simplexe et 5 pour la branche N3SNatur, c'est à dire alloués au calcul des champs fluides). Le tableau 4.1 donne pour chacun des calculs d'optimisation le temps de restitution nécessaire ainsi que le nombre d'itérations et le nombre de passage dans la branche N3SNatur.

CAS	Temps (h)	Nombre d'itérations	Nombre de Runs N3SNatur
1D Débit en B et C égaux	27	11	23
1D Cible impossible	19.5	8	16
1D Débit total donné	16.25	6	15
2D Profil symétrique	55	24	46
2D Profil asymétrique	35	18	35
3D Profil asymétrique	76.5	35	66

TAB. 4.1 – Résumé des runs présentés

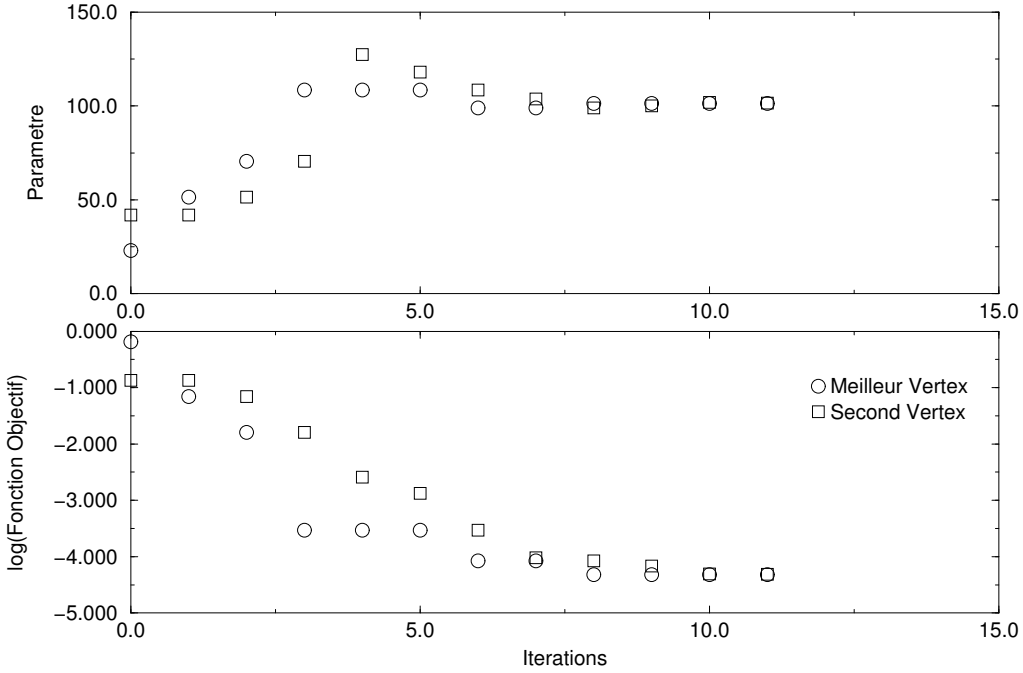


FIG. 4.4 – Optimisation 1D - Débit en B et C égaux : évolution du simplexe et de la fonction objectif en fonction des itérations

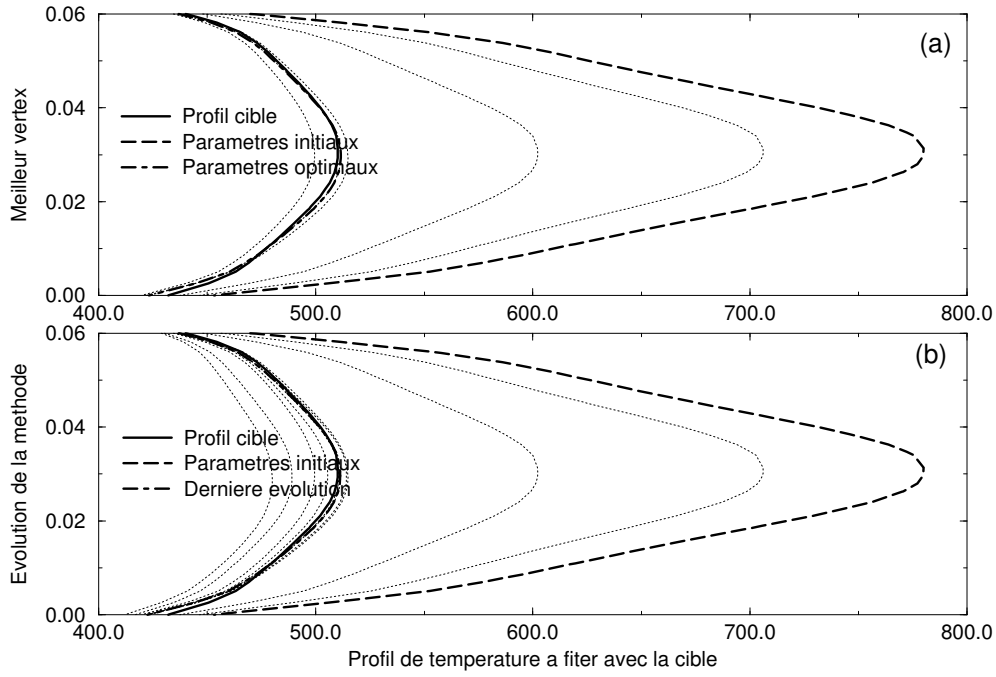


FIG. 4.5 – Optimisation 1D - Débit en B et C égaux : évolution des meilleurs profils (a) et des profils des itérés (b)

La fonction objectif de cas test est de forme parabolique avec un seul minimum sur l'espace d'état. L'algorithme parvient donc à converger rapidement vers la bonne solution.

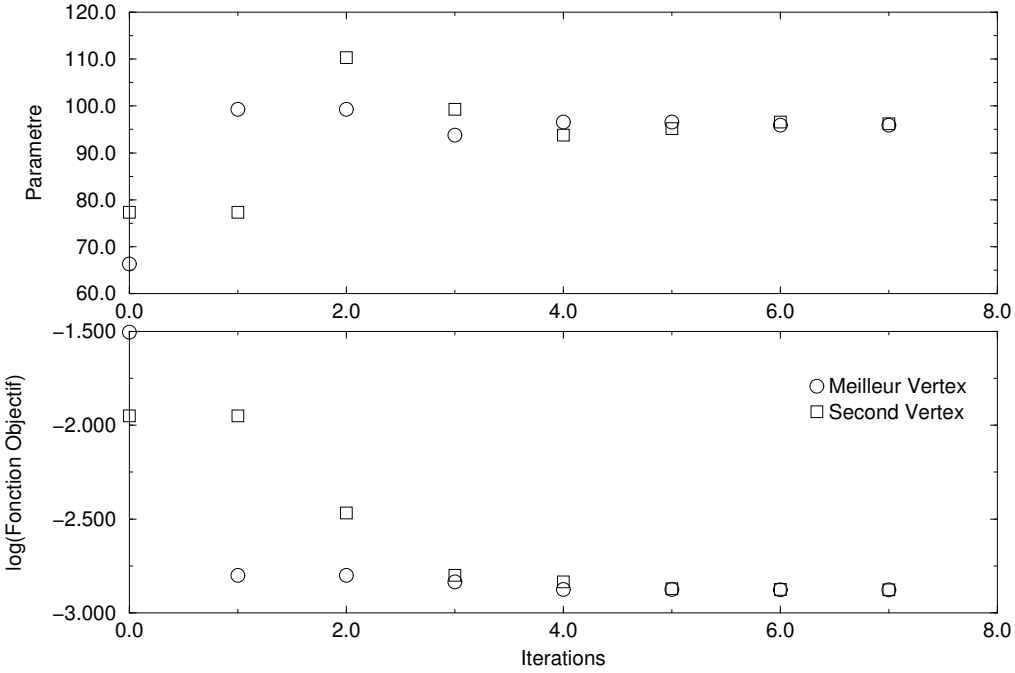


FIG. 4.6 – Optimisation 1D - Débit en B et C égaux avec une cible impossible à atteindre : évolution du simplexe et de la fonction objectif en fonction des itérations

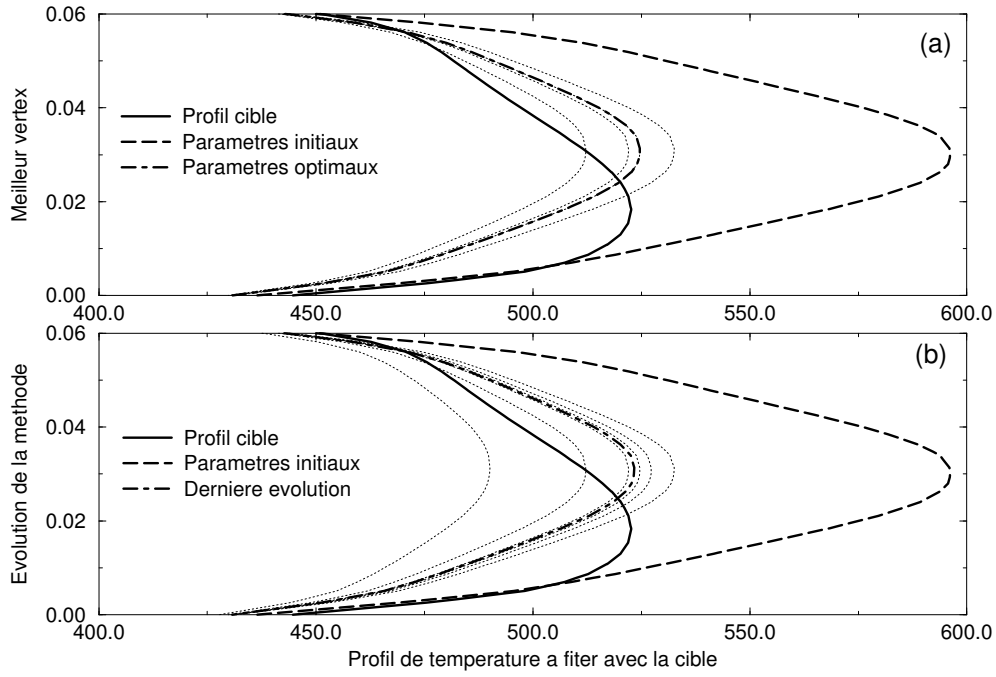


FIG. 4.7 – Optimisation 1D - Débit en B et C égaux avec une cible impossible à atteindre : évolution des meilleurs profils (a) et des profils des itérés (b)

Comme dans le cas précédent, la fonction objectif est simple et donc la convergence rapide. Du point de vue de la physique, l'algorithme donne comme optimum, une valeur de V_C qui permet d'avoir le même débit total de refroidissement ($\dot{m}_B + \dot{m}_C$) que celui obtenu avec la cible. La température T_D calculée analytiquement (Eq. D.3) est donc la même dans les deux cas.

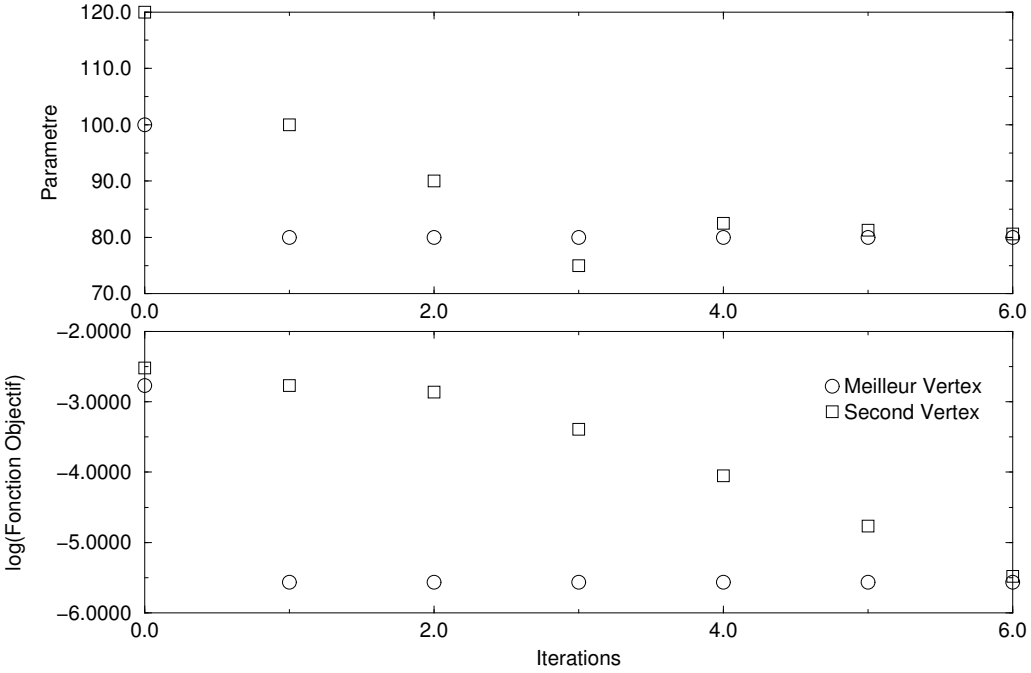


FIG. 4.8 – Optimisation 1D - Débit d'injection total donné : évolution du simplexe et de la fonction objectif en fonction des itérations

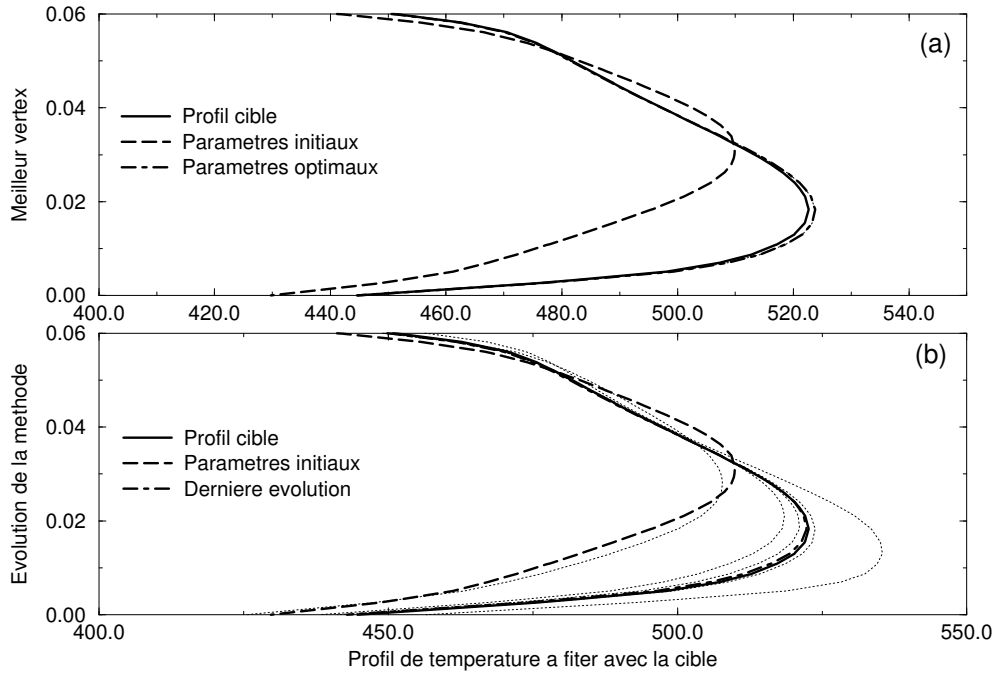


FIG. 4.9 – Optimisation 1D - Débit d'injection total donné : évolution des meilleurs profils (a) et des profils des itérés (b)

Une fois le simplexe initial bien positionné (FIG. 4.2), la convergence vers la solution optimale est rapide.

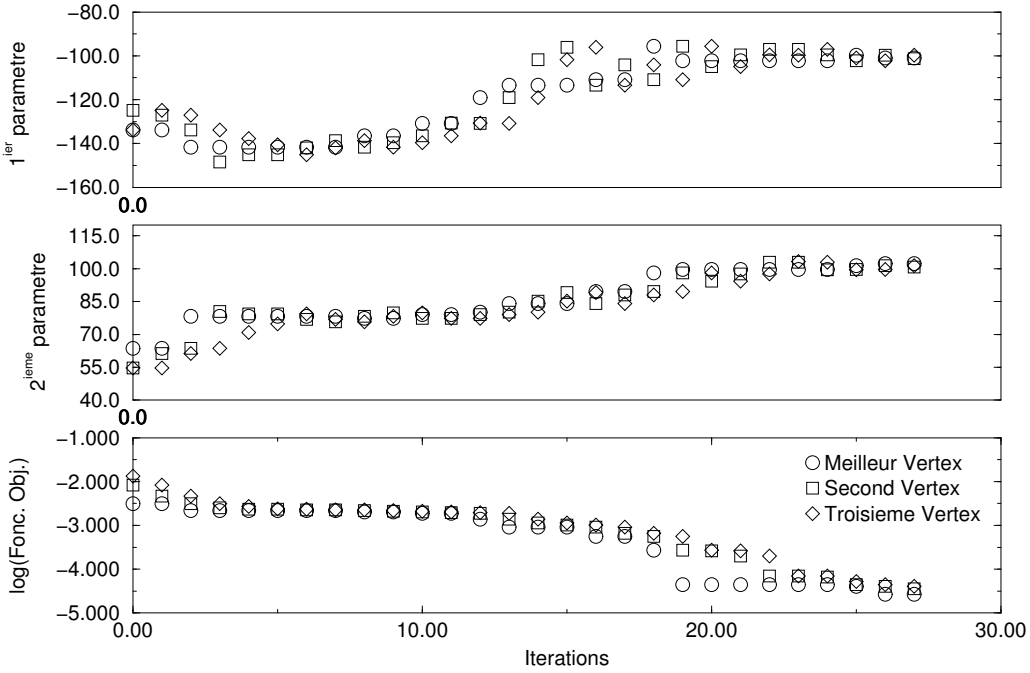


FIG. 4.10 – Optimisation 2D - Débit en B et C égaux : évolution du simplexe et de la fonction objectif en fonction des itérations

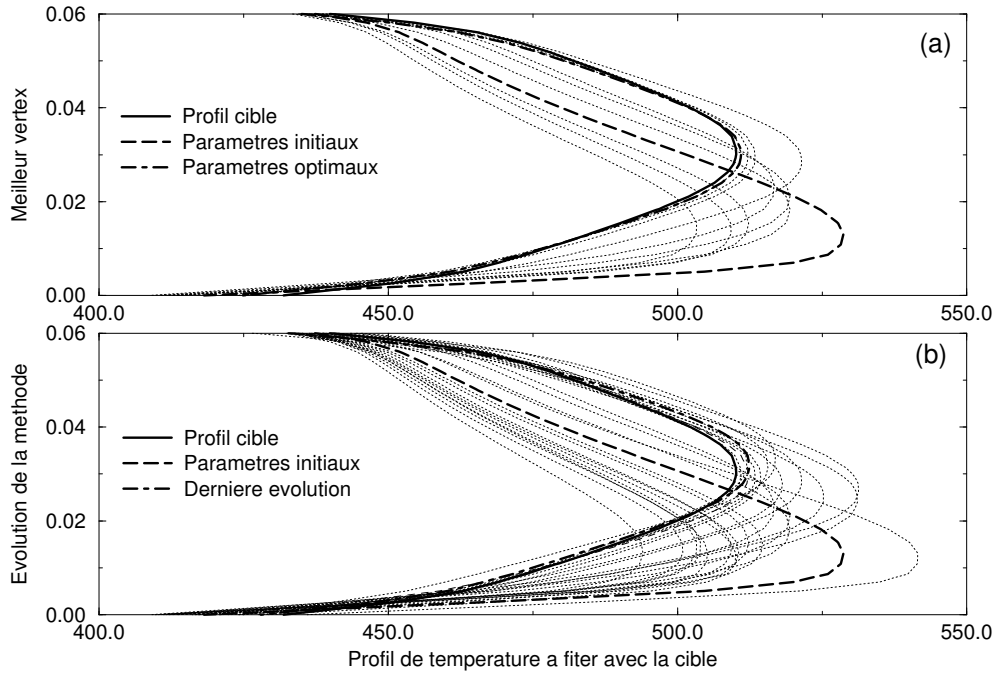


FIG. 4.11 – Optimisation 2D - Débit en B et C égaux : évolution des meilleurs profils (a) et des profils des itérés (b)

En augmentant le nombre de paramètre, on s'apperçoit que le nombre d'itération pour converger augmente aussi. La fonction objectif correspondant à ce cas, qui est symétrique par rapport aux paramètres d'optimisation, est relativement simple. Il n'existe q'un seul minimum et la convergence vers la cible est assurée quelque soit le simplexe de départ.

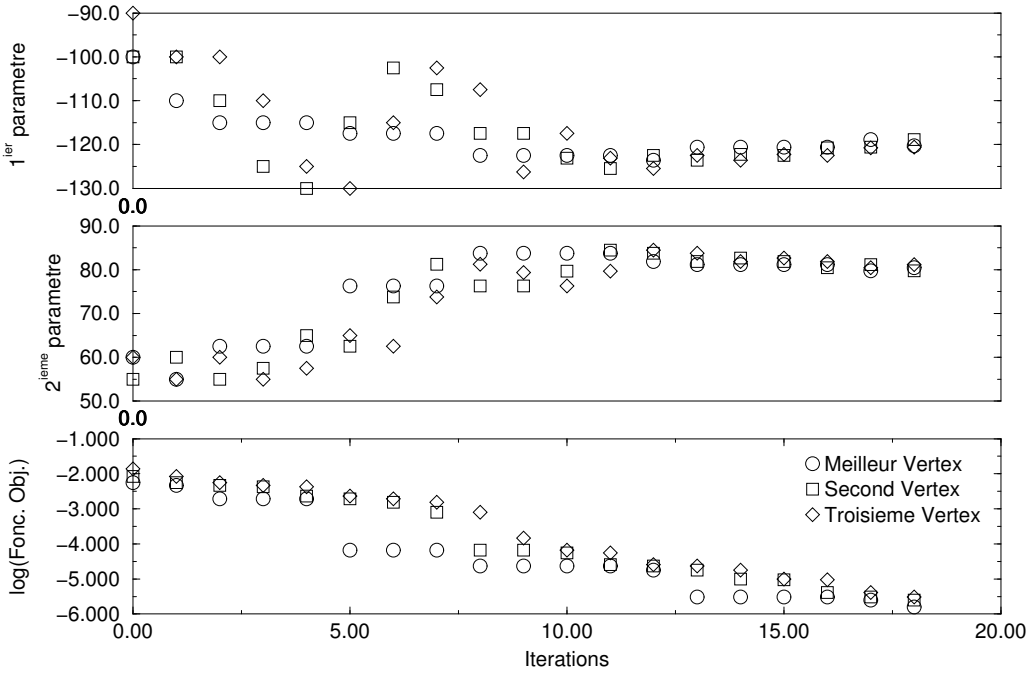


FIG. 4.12 – Optimisation 2D - Profil asymétrique : évolution du simplexe et de la fonction objectif en fonction des itérations

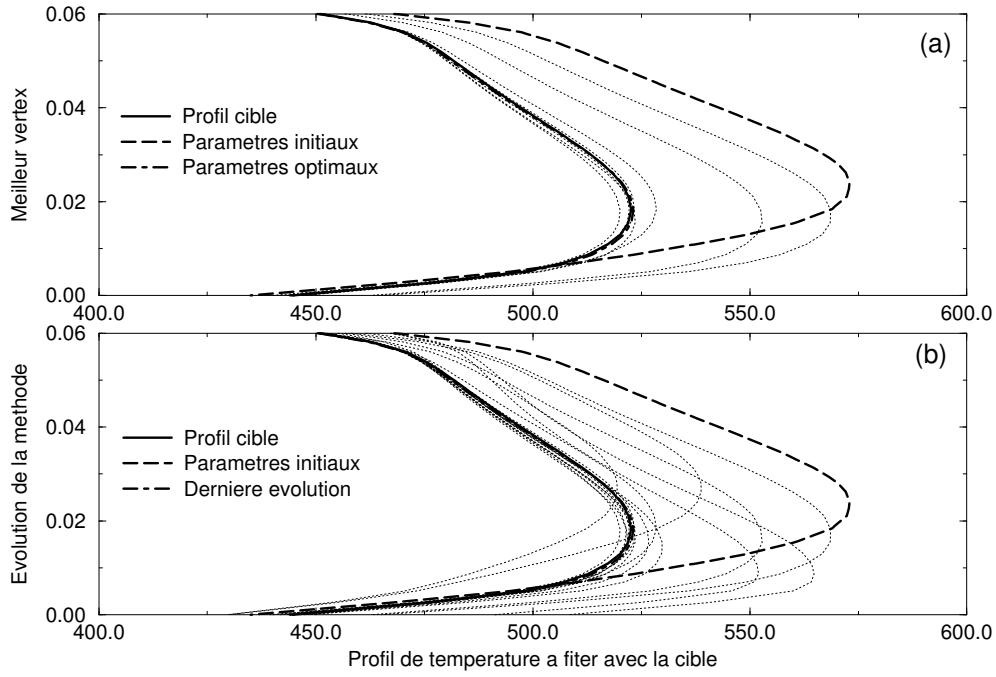


FIG. 4.13 – Optimisation 2D - Profil asymétrique : évolution des meilleurs profils (a) et des profils des itérés (b)

Le cas asymétrique fait intervenir plusieurs minimums locaux dans lesquels le simplexe est succeptible de s'enfermer. Le krigeage a donc été très utile pour comprendre comment le simplexe convergeait vers des extrémums locaux et pour l'initialiser de façon à être dans le bassin d'attraction du global.

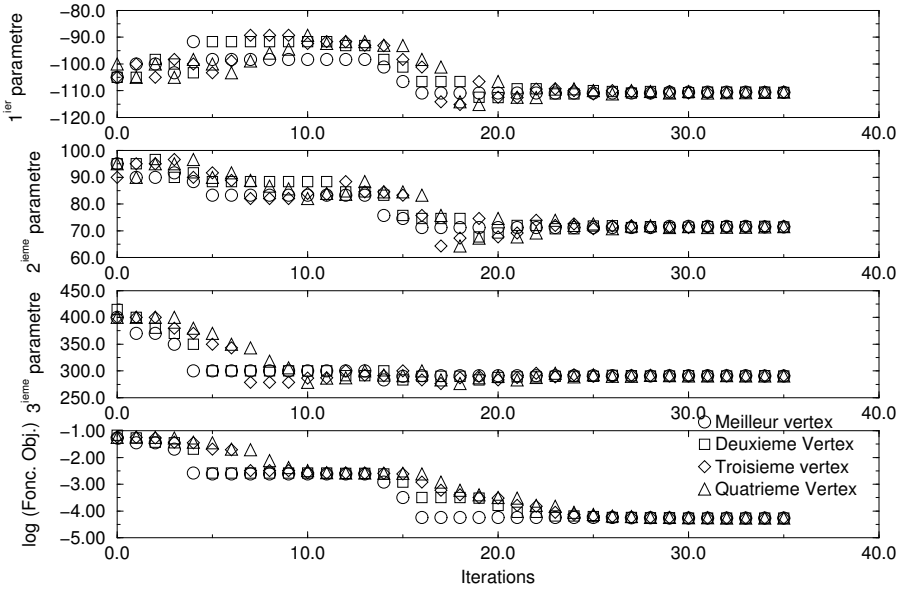


FIG. 4.14 – Optimisation 3D - Profil asymétrique : évolution du simplexe et de la fonction objectif en fonction des itérations

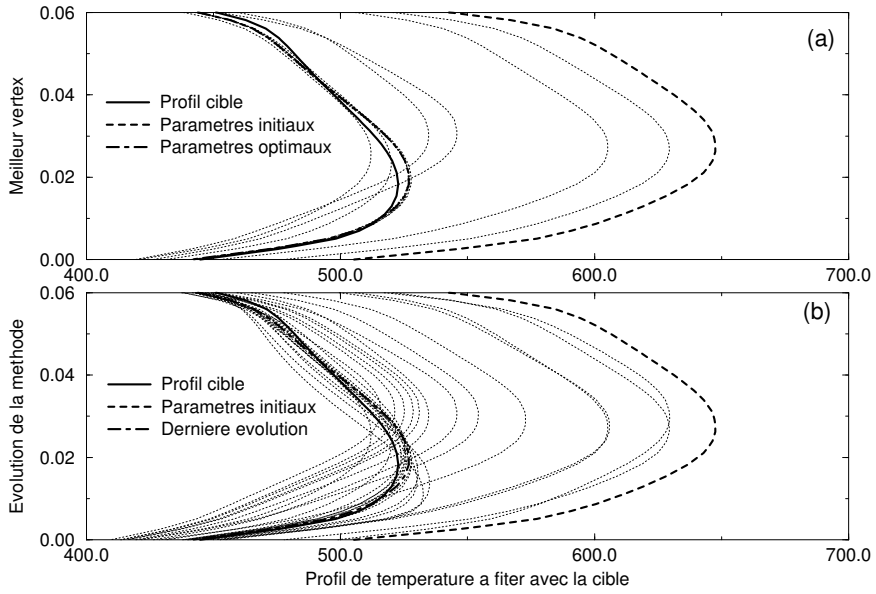


FIG. 4.15 – Optimisation 3D - Profil asymétrique : évolution des meilleurs profils (a) et des profils des itérés (b)

Ce cas test montre une fragilité de l'algorithme du simplexe : la méthode converge de façon prématurée. En effet, la convergence vers l'optimum semble se faire convenablement mais elle s'arrête. Il est important de remarquer la différence de rapidité de convergence des paramètres : le troisième paramètre (T_C) converge plus rapidement que les deux autres (vitesses V_B et V_C). La taille simplexe ne permet alors plus une exploration suffisante et il s'enferme dans un minimum local qui peut être purement numérique. Pour relancer la recherche, il faut réinitialiser le simplexe en gardant le meilleur vertex et placer les autres de façon à ce qu'il ait une taille suffisante. Notons que les valeurs moyennes des profils cible et optimisé sont identiques au demi Kelvin près : 498 K et que le calcul analytique (Eq. D.3) donne aussi le même résultat dans les deux cas : 483 K.

Conclusions et perspectives

L'objectif de ce travail était de tester les possibilités d'utiliser le code de *CFD* N3SNatur comme base pour la constitution d'une chaîne automatique d'optimisation. Pour cela, il a tout d'abord été nécessaire de parcourir la bibliographie existante sur le sujet. Ensuite, une prise en main du code N3SNatur et du coupleur PALM ont été réalisés. Enfin, les premiers pas vers la conception d'une boucle d'optimisation et la mise en place de cas tests ont montré la faisabilité du projet.

L'investigation bibliographique a permis de se rendre compte de ce qui se fait actuellement dans le domaine et surtout des principales difficultés que l'on peut y rencontrer. Ces difficultés sont liées non seulement aux méthodes d'optimisation en terme de performances dans leur cadre d'utilisation, mais aussi à leur couplage avec les codes de *CFD*, très gourmants en ressources informatiques.

L'algorithme du simplexe ayant été retenu pour sa facilité d'implantation et sa robustesse, le coupleur de code PALM a été utilisé pour la mise en oeuvre de l'outil d'optimisation. La partie la plus délicate aura été de rendre compatible le code N3SNatur avec le coupleur. Un certain nombre de tests ont ensuite permis de valider la technique d'intégration et de mettre en défaut l'algorithme d'optimisation utilisé. L'outil réalisé a encore du chemin à parcourir pour arriver à la maturité exigée par le projet européen INTELLECT D.M.

La modularité du coupleur utilisé aidera dans la suite du projet à intégrer l'unité N3SNatur dans un algorithme d'optimisation plus approprié aux demandes de TURBOMECA. Dans ce cadre, les études sur le choix de la fonction objectif seront à approfondir ainsi qu'une augmentation des échanges d'information entre l'algorithme et l'unité N3SNatur afin de limiter le temps de calcul lié à la détermination des champs fluides (connaissance de l'objectif par l'unité). Ensuite, un point primordiale est que l'outil existant permet de faire de l'optimisation sur les conditions aux limites sans toucher à la géométrie. Il était indispensable de se consacrer uniquement sur cet aspect pour valider la faisabilité du projet. Cependant, une étape importante est d'intégrer la possibilité d'optimisation de forme. Ceci inclut des modifications sur le maillage qui peuvent être faites soit par le remaillage automatique du domaine (demandé par TURBOMECA mais très difficile et peu implanté en pratique) soit par la déformation du maillage existant (garder la topologie en changeant les coordonnées des points, ce qui est très utilisé notamment en couplage multi physique de type fluide-structure). Enfin, il sera indispensable de faire fonctionner l'outil sur divers plateformes, la seule utilisée pour le moment étant une machine SGI Origin 2000 du CERFACS.

La suite de ces travaux sera réalisée dans le cadre d'une thèse qui débutera en Octobre 2004 au CERFACS.

Bibliographie

[1] D. Abramson, A. Lewis, T. Peachey, and C. Flecher. An automatic design optimization tool and its application to computational fluid dynamics. *Communication of the ACM*, 2001.

[2] D. Bueche, P. Stoll, R. Dornberger, and P. Koumoutsakos. Multi-objective evolutionary algorithm for the optimization of noisy combustion problems. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 32 :460–473, 2002.

[3] J.C. Culioli. *Introduction à l'optimisation.* Ellipses, 1994.

[4] R. Dornberger, P. Stoll, D. Bueche, and A. Neu. Multidisciplinary turbomachinery blade design optimization. *AIAA*, 2000-0838, 2000.

[5] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[6] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Professional, 1989.

[7] A.P. Gurson. *Simplex Search Behavior in Noneminear Optimization.* PhD thesis, College of William & Mary in Virginia, 2000.

[8] T. Hiroyasu, M. Miki, J. Kamiura, S. Watanabe, and H. Hiroyasu. Multi-ojective optimization of diesel engine emissions and fuel economy using genetic algorithms and phenomenological model. Technical report, Society of Automotive Engeneers, Inc., 2002.

[9] R. Hooke and T.A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8 :212–229, 1961.

[10] S. Izquierdo. *Reducción de contaminantes en combustión aplicando técnicas de CFD y Algoritmos Genéticos.* PhD thesis, University of Zaragora, 2003.

[11] S. Kern, S.D. Muller, N. Hansen, D. Bueche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3 :77–112, 2004.

[12] B. Labeyrie. *Une méthode de krigeage pour l'optimisation multi-critère en calculs aérodynamiques.* ENSEEIHT, 2004.

[13] R. Martin. *N3S-NATUR V1.4, Manuel Theorique.* SIMULOG, 2001.

[14] R. Martin. *N3S-NATUR V1.4, Manuel Utilisateur.* SIMULOG, 2002.

[15] K. I. M. McKinnon. Convergence of the nelder-mead simplex method to a nonstationary point. *SIAM Journal of optimization*, 9 :148–158, 1998.

[16] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids.* Oxford Science Publications, 2001.

[17] F. Muyl, L. Dumas and V. Herbert. Hybrid method for aerodynamic shape optimization in automotive industry. *Computers and Fluids*, 33 :849–858, 2004.

[18] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7 :308–313, 1965.

[19] T. Ray and H. M. Tsai. Swarm algorithm for single and multiobjective airfoil design optimization. *AIAA*, 42, February 2004.

[20] G.R. Spendley, W. Hext and F.R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4 :148–158, 1962.

[21] PALM Team. *PALM and PREPALM Research QuickStart Guide*. CERFACS, 2001.

[22] V. Torczon. *Multi Directionnal Seach : A Direct Seach Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, 1989.

ANNEXES

Annexe A

Tests sur l'algorithme de Nelder & Mead

Un certain nombre de tests ont été réalisés dans le but de valider l'écriture du code et d'en constater les limites. Dans cette annexe, les fonctions utilisées pour les tests sont tout d'abord présentées. Ensuite, la convergence de l'algorithme vers des extremums est discutée. Enfin, l'algorithme de Nelder et Mead est comparé avec une autre méthode de type simplexe.

Fonctions tests utilisées

Cinq fonctions ont été utilisées pour mener les tests sur l'algorithme de Nelder & Mead. Chacune d'elles a des spécificités précises qui ont permis de valider le bon comportement de la méthode. Ces fonctions sont :

- la fonction carré
- la fonction de Monge Ampère
- la fonction de McKinnon
- la fonction de Rosenbrock
- la fonction de Dornberger

Notons que toutes ces fonctions ont été volontairement choisies à deux paramètres. Le code a été écrit de façon à ce que le nombre de variables d'optimisation soit une entrée manipulable par l'utilisateur. Des tests dans des espaces de recherche d'ordre plus élevé sont à envisager dans le futur.

La fonction carré

Cette fonction est basique, ne possédant qu'un seul minimum mais permet de debugger le code rapidement. Son expression est :

$$\begin{array}{l} f(x,y) = x^2 + y^2 \\ (x,y) \in [-10,10]^2 \\ \text{minimum en } (x,y) = (0,0) \end{array} \tag{A.1}$$

Le but de son utilisation est de voir si l'algorithme converge bien vers un minimum sur une fonction simple.

La fonction de Monge Ampère

L'interêt de cette fonction basée sur de la trigonométrie est qu'elle possède plusieurs minimums qui sont égaux. Elle permet de voir comment se comporte le simplexe initialisé dans des bassins d'attraction. Son expression est donnée par :

$$\begin{array}{l} f(x,y) = sin(\pi x) * sin(2\pi y) \\ (x,y) \in [-2,2] \times [-1,1] \end{array} \tag{A.2}$$

La fonction de McKinnon

Un brusque changement de pente proche du minimum caractérise cette fonction. Ainsi, il possible de vérifier que la recherche s'arrête bien au minimum quelque soit la pente sur laquelle le simplexe est "posé" initialement. C'est sur cette fonction que McKinnon [15] à montré que l'algorithme du simplexe pouvait ne pas converger vers le minimum.

$$\begin{array}{l} f(x,y) = 360x^2 + y + y^2 \text{ si } x < 0 \\ f(x,y) = 6x^2 + y + y^2 \text{ si } x \geq 0 \\ (x,y) \in [-10,10]^2 \\ \text{minimum en} (x,y) = (0,-0.5) \end{array} \tag{A.3}$$

La fonction de Rosenbrock

Cette fonction est très utilisée pour tester la rapidité de convergence des algorithmes d'optimisation. En effet, une de ses particularités, qui lui faut son surnom "banana", est d'avoir une vallée en forme de banane dans lequel se situe le minimum. Autour de cette vallée, la pente (le gradient) de la fonction est important par rapport a celui à l'intérieur, où se trouve le minimum.

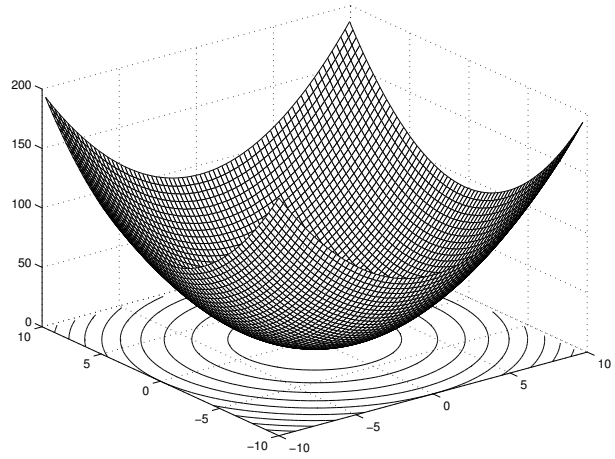
$$\begin{array}{l} f(x,y) = 100(y - x^2)^2 + (1-x)^2 \\ (x,y) \in [-1.5,1.5] \times [-1,2] \\ \text{minimum en } (x,y) = (1,1) \end{array} \tag{A.4}$$

La fonction de Dornberger

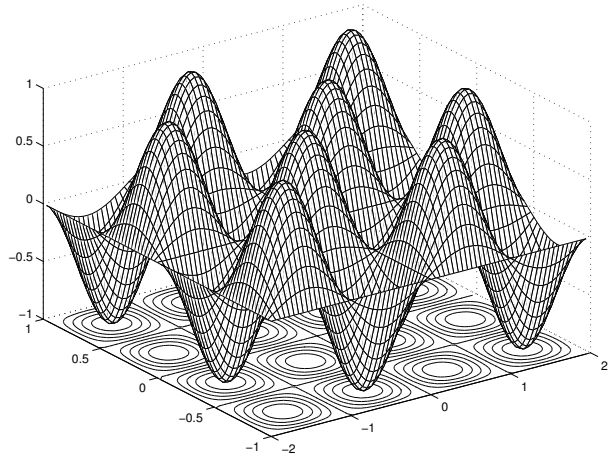
Pour finir, cette fonction a été choisie car elle est multi-modale : elle présente plusieurs minimums d'intensité differente. Il y a donc des minimums locaux et un minimum global sur l'espace d'état considéré. Elle se rapporte plus à des cas que l'on peut rencontrer en couplage d'optimisation et de CFD. Elle permet de constater plus précisément le comportement du simplexe soumis à la présence de bassins d'attractions divers et la bonne convergence de l'algorithme dans cette situation. Son expression mathématique est donnée par :

$$\begin{array}{l} f(x,y) = 3(1-x)^2 e^{\left(-x^2-(y+1)^2\right)} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{\left(-x^2-y^2\right)} - e^{\left(\frac{-(x+1)^2-y^2}{3}\right)} \\ (x,y) \in [-2,2] \times [-2.5,2.5] \end{array} \tag{A.5}$$

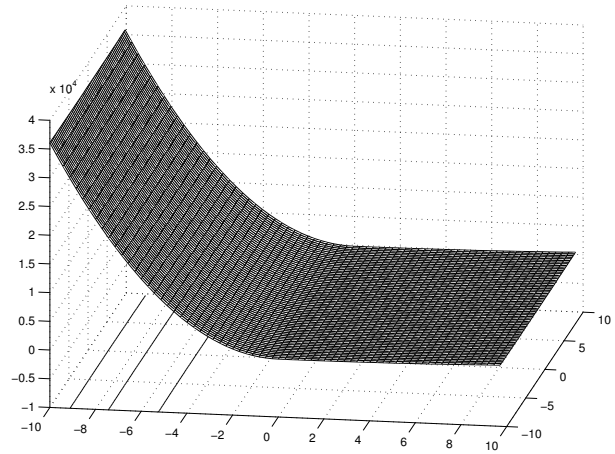
La figure A.1 donne la représentation graphique de ces fonctions tests.



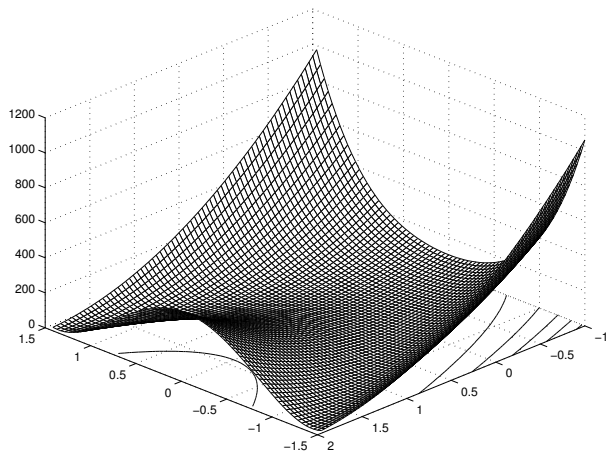
Fonction carré



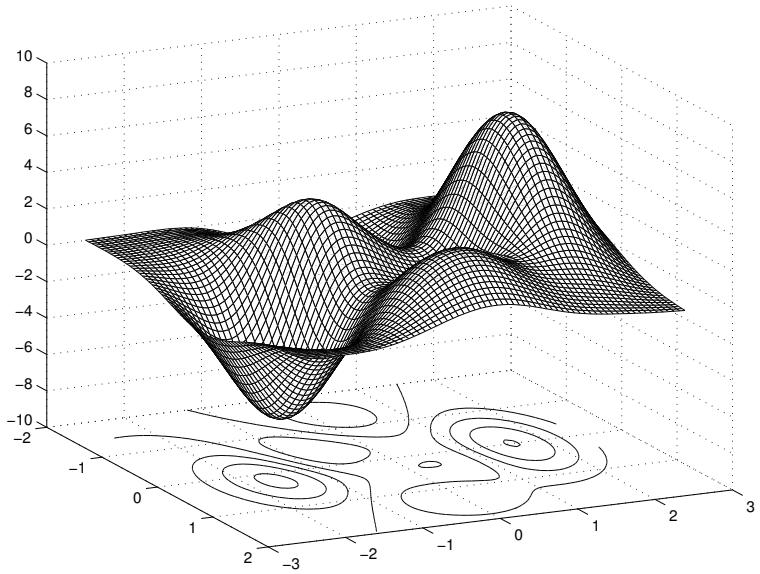
Fonction de Monge Ampère



Fonction de McKinnon



Fonction de Rosenbrock



Fonction de Dornberger

FIG. A.1 – Allure des fonctions tests sur leur espace d'état

Convergence de l'algorithme de Nelder & Mead

Initialisation du simplexe

Afin de tester la bonne convergence du simplexe vers le ou les minimums de l'espace d'état, 100 runs sont effectués pour chaque fonction. Le simplexe est initialisé de façon aléatoire pour chaque run de la manière suivante (FIG. A.2) :

- un premier vertex est tiré aléatoirement dans les bornes de l'espace de recherche (V1)
- pour les deux autres sommets (*i.e.* en optimisation à $n = 2$, trois vertex sont nécessaires pour constituer un simplexe), un incrément sur chaque direction est donné par rapport au premier vertex : selon x pour V2 et selon y pour V3

Le simplexe ainsi obtenu est dit orthogonal et possède de bonnes propriétés de recherche de minimum car il est non dégénéré et permet une approximation de gradient optimale dans cette configuration.

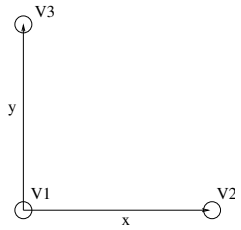


FIG. A.2 – Initialisation du simplexe

Le critère de convergence utilisé dans ces simulations est de $\epsilon = 10^{-8}$ (*cf.* Eq. 2.1). Cette section étant dédiée à l'étude de la convergence, il ne sera pas présenté de graphique d'évolution de la fonction coût versus les itérations ni de nombre d'itérations versus les runs.

Etude de la convergence du simplexe

Pour représenter la convergence de l'algorithme sur les diverses fonctions tests retenues, le principe suivant a été adopté : des isocontours de la fonction objectif sont tracés ainsi que les 100 vertex V1 choisis aléatoirement (croix) et les points de convergence (cercles). Les résultats sont montrés sur les figures A.3.

On remarque dans chacun des cas une convergence des simplexes vers les minimums attendus. Dans les cas de Monge Ampère et Dornberger, on constate l'effet des bords de l'espace de recherche : des simplexes ont convergé vers les frontières qui, vu le traitement qui leur est fait (valeur de la fonction objectif très grande si un vertex sort du domaine) peuvent être des localisations de minimums.

Concernant le cas de la fonction de McKinnon, il est possible de montrer [15] que pour le simplexe de départ défini par

$$s = \begin{pmatrix} 0,0 \\ \frac{1+\sqrt{33}}{8}, \frac{1-\sqrt{33}}{8} \\ 1,1 \end{pmatrix} \tag{A.6}$$

la convergence tend à donner comme solution le point (0,0). En effet, les opérations acceptées à chaque itération sont des contractions internes des sommets autres que (0,0). Dans la plupart des cas, on se rend donc compte que le simplexe convergera vers les bonnes solutions. Il apparait toutefois nécessaire de connaître de façon grossière la typologie de la fonction que l'on étudie

(*i.e.* les bassins d'attraction) afin de positionner le simplexe initial de la meilleure manière qu'il soit (la taille du simplexe doit notament être inférieure à celle du bassin d'attraction).

Comparaison de l'algorithme de Nelder & Mead avec un simplexe multi-directionnel

Nous avons vu au chapitre 2 qu'un des principaux défauts de l'algorithme du simplexe est lié à la dégénérescence de la figure géométrique qui peut tendre avec les opérations à ne plus être une base de l'espace de recherche. Torczon [22] a proposé une version de l'algorithme qui palie à cet aspect. Il s'agit d'une version qu'elle a nommée "multi directionnelle" car les transformations sont appliquées à tous les vertex sauf le meilleur qui a le rôle de centroide. Par nature, cet algorithme est parallèle puisque tous les vertex sont soumis en même temps aux changements. De plus, il conserve la forme initiale du simplexe, seule sa taille peut changer. Ainsi, il ne dégénère pas.

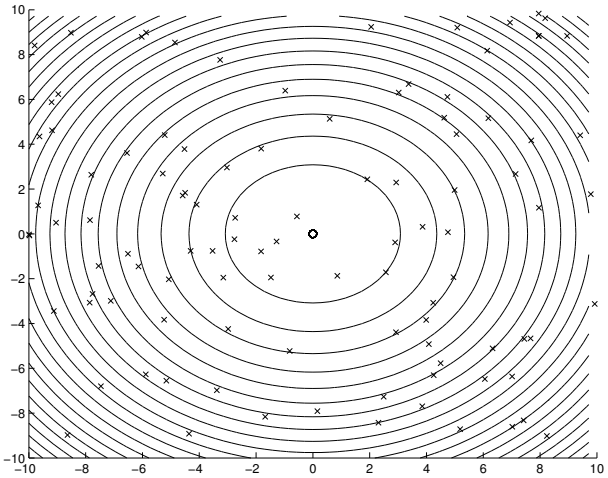
Les deux algorithmes ont été comparés sur les fonctions carré (fonction simple servant de référence), de Rosenbrock (où la convergence avec l'algorithme de Nelder & Mead est très lente) et de Dornberger (pour avoir une fonction réaliste).

Dans chacun des cas, on compare pour la même séquence de 100 simplexes initiaux aléatoires :
- les points de convergence (FIG. A.4)
- le nombre d'itérations nécessaires à la convergence (FIG. A.5)
- le nombre d'évaluation de la fonction objectif (FIG. A.5)

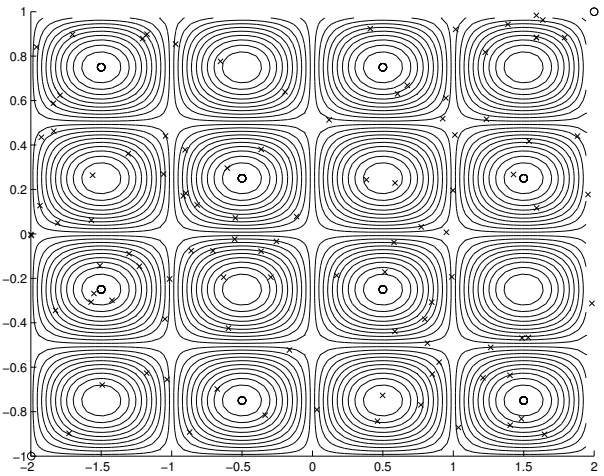
Concernant les points de convergence, les figures A.4 montrent que l'algorithme de Torczon est plus précis. Notamment, dans le cas de la fonction de Rosenbrock, on constate une convergence prématurée de l'algorithme de Nelder & Mead, liée directement aux déformations du simplexe.

La contrepartie de cette précision est visible sur les figures A.5 qui représentent le nombre d'itérations et le nombre d'appel à la routine de calcul de la fonction objectif. Même si, dans le cas des fonctions carré et de Dornberger, le nombre d'itérations est moins important en multi directionnel qu'en mono directionnel, le nombre d'évaluations de la fonction objectif (sauf accident) est supérieur en multi directionnel. Or n'oublions pas que ce qui coûte cher dans l'optimisation en CFD, c'est le calcul de cette fonction. Pour la fonction de Rosenbrock, le fait que le simplexe ne se déforme pas durant la méthode multi directionnelle est très pénalisant : les nombres d'itérations et d'appel à la routine de calcul de fonction objectif deviennent démesurément grand devant le cas mono directionnel.

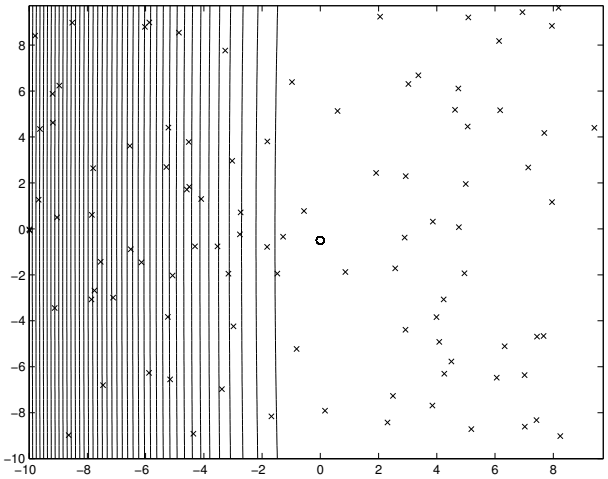
A la vue de ces résultats, on comprend l'interêt porté à l'algorithme de Nelder & Mead dans une phase de première approche de couplage entre l'optimisation et la CFD. Toutefois, il est envisagé de faire évoluer cet algorithme (en le couplant avec d'autres méthode de type gradient par exemple) ou de le changer pour un meilleur (BFGS) dans le cadre des applications à résoudre.



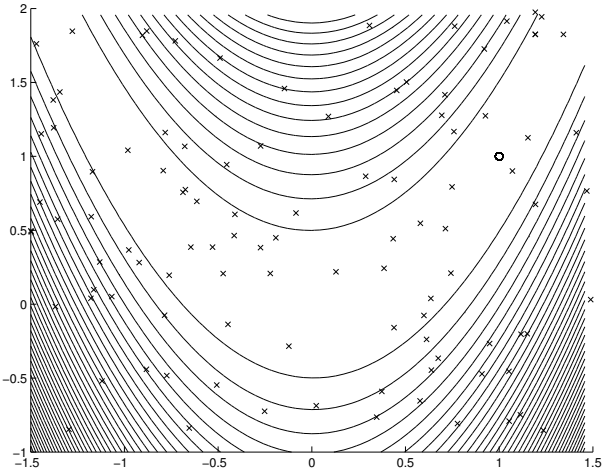
Fonction carré



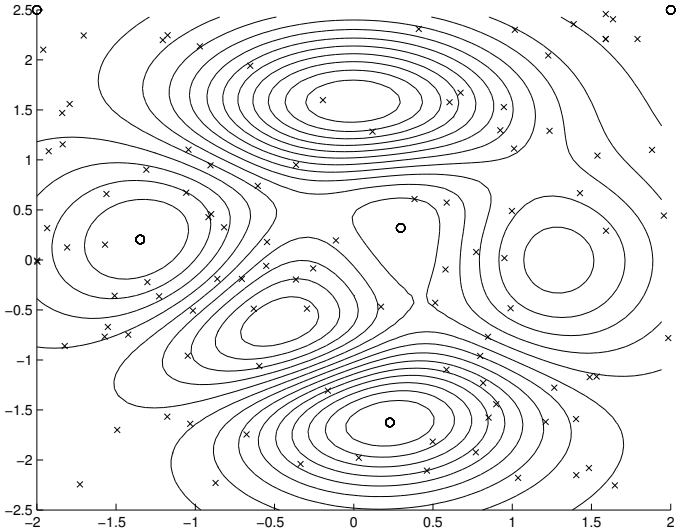
Fonction de Monge Ampère



Fonction de McKinnon

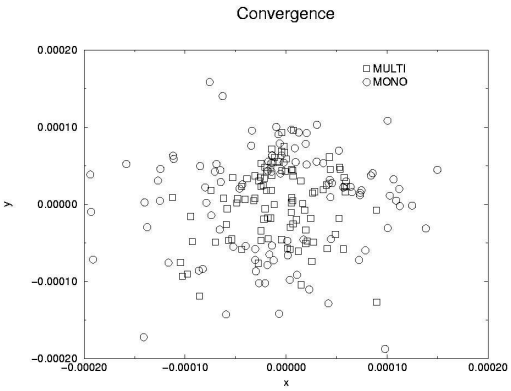


Fonction de Rosenbrock

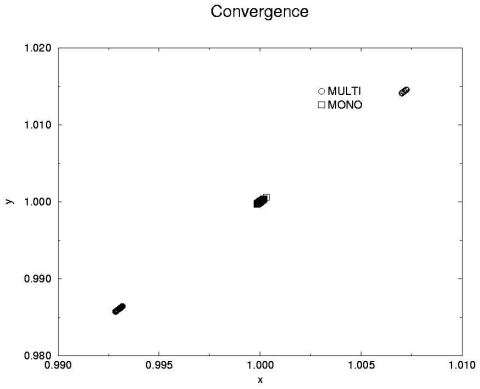


Fonction de Dornberger

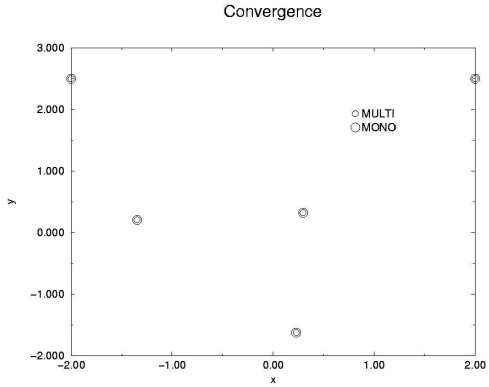
FIG. A.3 – Tracés des points de convergence



Fonction carré

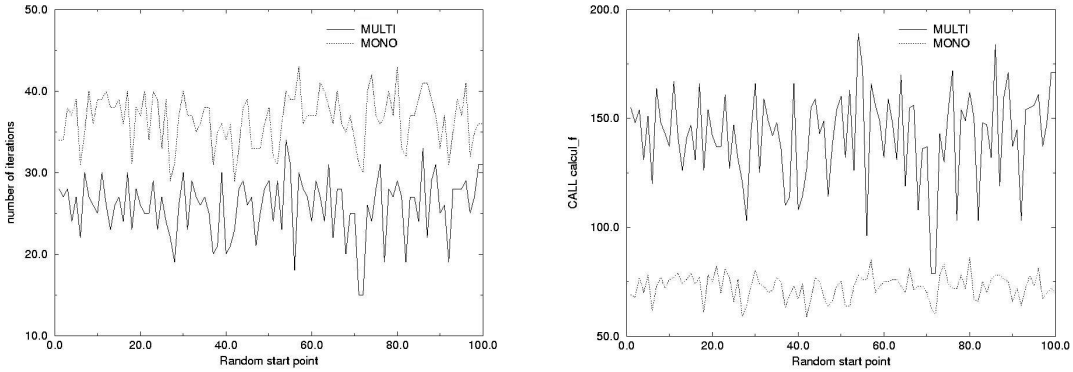


Fonction de Rosenbrock

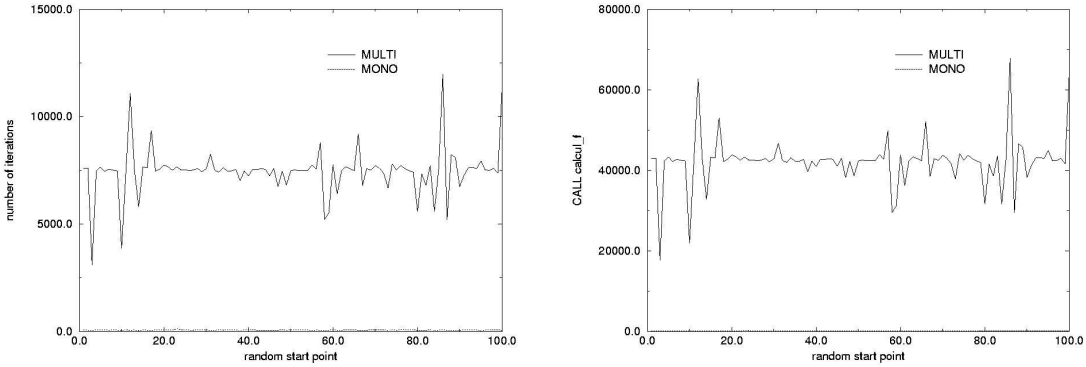


Fonction de Dornberger

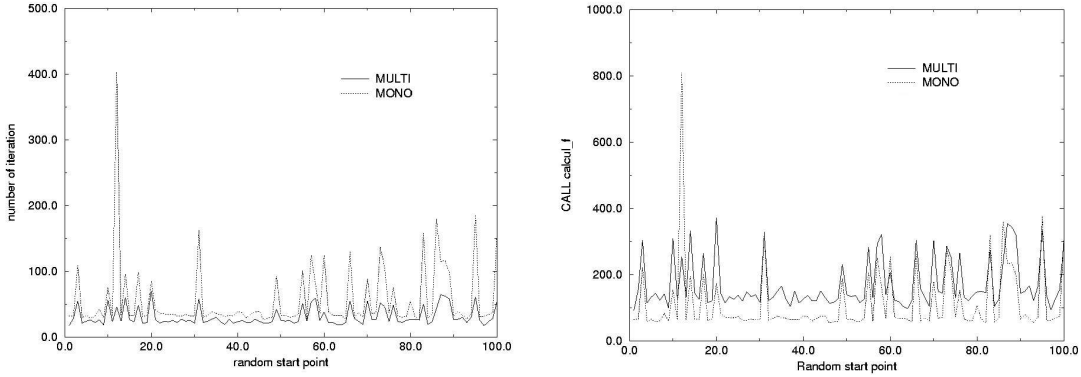
FIG. A.4 – Points de convergence des fonctions : carré, Rosenbrock et Dornberger



Fonction carré



Fonction de Rosenbrock



Fonction de Dornberger

FIG. A.5 – Nombre d'itérations et d'appel à la routine de calcul de la fonction ojectif dans les cas : carré, Rosenbrock et Dornberger pour les 100 initialisations aléatoires de simplexe

Annexe B

Détails sur les modifications à faire pour insérer N3SNatur dans PALM

Cette annexe présente les modifications à réaliser dans le code N3SNatur pour le rendre compatible avec le coupleur de code PALM. Il sera tout d'abord exposé les modifications standard de code (qui sont celles à réaliser quelque soit le code que l'on veut insérer dans PALM), puis celles liées directement à la programmation initiale de N3SNatur.

Organisation des sources N3SNatur

Les sources du code N3SNatur ont été livrées au CERFACS selon la configuration présentée sur la figure B.1. Elles sont compilables sur différentes architectures mais le projet a été entièrement réalisé sous SGI 64 bits (d'où les références à SGIMP64 que l'on pourra trouver dans cette annexe).

Les dossiers qui contiennent les sources sont NATURC/ et noyau/ pour le Fortran 77 et noyau90/ pour le Fortran 90.

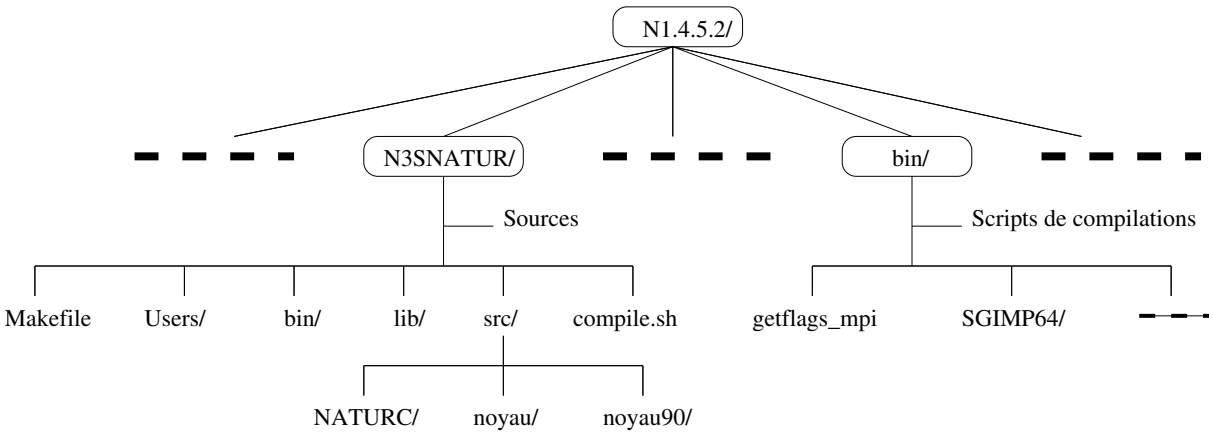


FIG. B.1 – Organisation des sources N3SNatur

Les modifications standard

La carte d'identité de l'unité N3SNatur

Le fichier qui contient le programme principal parallèle est "N3SNATUR/src/noyau90/s90/n3snatur_s.F". Dans ce fichier il est nécessaire de remplacer *PROGRAM* par *SUBROUTINE*, car une unité est considérée comme une subroutine pour PALM. De plus, pour connaître les unités et charger une interface de communication, l'utilisateur doit écrire une carte d'identité de l'unité (TAB. B.1). On y trouve le nom de l'unité ainsi que des informations concernant les échanges envisagés entre l'unité et l'environnement dans l'algorithme de couplage (Voir le manuel de PALM [21] pour plus de détails).

PALM_UNIT	-sub n3snatur_s
	-comment {Serveur N3SNATUR pour le calcul parallele}

TAB. B.1 – Carte d'identité de l'unité N3SNatur

Initialisation et destruction de processus

Le parallélisme étant géré par le coupleur, il est dangereux de garder dans le code à insérer dans PALM des primitives MPI qui servent à l'initialisation ou à la destruction de processus. Il est donc conseillé de retirer les instructions MPI_INIT et MPI_FINALIZE et de remplacer MPI_ABORT par l'instruction interne de PALM : PALM_ABORT. Le tableau B.2 montre les subroutines dans lequelles on trouve ces instructions dans le code. Il est à noter que ces instructions ont été, dans le cadre de cette étude, modifiées à la main.

Primitives	*Subroutines concernées*
MPI_INIT	N3SNATUR/src/noyau90/s90/n3snatur_s.F
MPI_FINALZE	N3SNATUR/src/noyau90/COUPLAGE/fin_proc.F N3SNATUR/src/noyau90/PARALL/endcom.F
MPI_ABORT	N3SNATUR/src/noyau90/COUPLAGE/fin_proc.F N3SNATUR/src/noyau90/PARALL/launchnode.F

TAB. B.2 – Primitives à supprimer ou à changer pour insérer un code dans PALM

Le communicateur PL_COMM_EXEC

Les communications en MPI sont facilitées par l'existence des communicateurs et des groupes (voir le chapitre 3). Un communicateur regroupe un certain nombre de processus qui peuvent ainsi se connaître et partager des informations, se synchroniser ... Le communicateur de base qui regroupe tous les processus à l'initialisation de MPI est MPI_COMM_WORLD. Dans le cadre du couplage de code, MPI est initialisé par PALM et donc ce communicateur regroupe tous les processus de l'application. Un nouveau communicateur doit donc être mis en place pour regrouper les processes du code à intégrer. PALM propose un communicateur prédéfini pour les codes parallèles : PL_COMM_EXEC. Il est alors nécessaire de remplacer dans le code source tous les MPI_COMM_WORLD par des PL_COMM_EXEC.

Cette modification devant être faite dans 47 routines (TAB B.3), un script écrit en langage SHELL permet de le faire de façon automatique. Précisons que, comme dans le cas de l'appel de

la primitive PALM_ABORT, il est nécessaire d'ajouter en en-tête des routines un "use palmlib" qui est une librairie de PALM.

Les modifications liées à N3SNatur

Au delà des modifications standard que l'on doit faire quelque soit le code à intégrer dans PALM, N3SNatur offre un certain nombre de spécificités qui ont compliqué cette tâche.

Restitution de tous les processeurs

Le principe général du parallélisme de N3SNatur, est d'avoir un processus maître qui initialise les processus esclaves et met fin à l'application, ainsi que des processus esclaves qui réalisent les calculs (résolution des équations de la Mécanique des Fluides). Concernant N3SNatur sans couplage, le code est unique : un seul exécutable est généré lors de la compilation des sources. Les processeurs savent les instructions qu'ils doivent exécuter par rapport à leur rang dans le communicateur MPI_COMM_WORLD (0 pour le maître). C'est ainsi que dans la routine "N3SNATUR/src/noyau90PARALL/launchnode.F", les processus esclaves sont stoppés à la fin du run par l'instruction *stop*. De ce fait, lors du couplage, les processeurs ne sont pas "rendus" à PALM après un passage dans l'unité N3SNatur. Il faut donc impérativement contourner cette instruction.

Désallocation des tableaux

Dans les sources de N3SNatur écrites en Fortran 90, il existe un grand nombre de tableaux qui sont alloués en mémoire de manière dynamique. Ces tableaux passent d'une routine à l'autre par le biais des modules (langage de Fortran 90). Ces tableaux n'étant pas désalloués à la fin du code, lorsque l'unité N3SNatur est appelée pour une deuxième fois dans PALM, l'allocation pour ces même tableaux est "redemandée". L'exécution s'arrête car cette opération n'est absolument pas permise.

Un script SHELL permet de localiser dans tous les modules de N3SNatur les tableaux alloués et écrit une routine de désallocation des tableaux sur le principe suivant :

IF ALLOCATED (TABLEAU) DEALLOCATE (TABLEAU)

Cette routine est alors appelée en fin d'unité N3SNatur dans la routine n3snatur_s.F (anciennement programme principal). Bien que brutale, cette méthode s'avère être efficace.

Autres détails

Deux autres modifications sont à signaler :

- dans la routine "N3SNATUR/src/noyau90/PARALL/launchnode.F", il est nécessaire d' ajouter une instruction *RETURN* visible par les esclaves (rappelons que c'est dans cette routine que les esclaves étaient stoppés). En effet, cette instruction existe mais elle concerne uniquement le maître.
- il existe un bug non résolu de PALM concernant le nombre maximum de groupes utilisables par MPI : la variable d'environnement MPI_GROUP_ MAX doit être fixée à une grande valeur.

N3SNATUR/src/noyau90/COUPLAGE/envoi_info.F
N3SNATUR/src/noyau90/COUPLAGE/fin_proc.F
N3SNATUR/src/noyau90/COUPLAGE/rec_header.F
N3SNATUR/src/noyau90/COUPLAGE/receive_eFs_header.F
N3SNATUR/src/noyau90/COUPLAGE/receive_eFs_sig.F
N3SNATUR/src/noyau90/COUPLAGE/receive_eFs_sol.F
N3SNATUR/src/noyau90/COUPLAGE/receive_eFs_tsource.F
N3SNATUR/src/noyau90/COUPLAGE/receive_sFe_sig_ts.F
N3SNATUR/src/noyau90/COUPLAGE/receive_sFe_sol.F
N3SNATUR/src/noyau90/COUPLAGE/reception.F
N3SNATUR/src/noyau90/COUPLAGE/recevoir_info.F
N3SNATUR/src/noyau90/COUPLAGE/send_e2s_header.F
N3SNATUR/src/noyau90/COUPLAGE/send_e2s_sig.F
N3SNATUR/src/noyau90/COUPLAGE/send_e2s_sig_ts.F
N3SNATUR/src/noyau90/COUPLAGE/send_e2s_sol.F
N3SNATUR/src/noyau90/COUPLAGE/send_s2e_header.F
N3SNATUR/src/noyau90/COUPLAGE/send_s2e_sig.F
N3SNATUR/src/noyau90/COUPLAGE/send_s2e_sol.F
N3SNATUR/src/noyau90/COUPLAGE/send_s2e_tsource.F
N3SNATUR/src/noyau90/PARALL/arecwn.F
N3SNATUR/src/noyau90/PARALL/asedwn.F
N3SNATUR/src/noyau90/PARALL/endcom.F
N3SNATUR/src/noyau90/PARALL/launchnode.F
N3SNATUR/src/noyau90/PARALL/reclaunch.F
N3SNATUR/src/noyau90/PARALL/releaseall.F
N3SNATUR/src/noyau90/PARALL/test_rec_mess.F
N3SNATUR/src/noyau90/TURBO/rec_turb_eFs_ini.F
N3SNATUR/src/noyau90/TURBO/rec_turb_eFs_maj.F
N3SNATUR/src/noyau90/TURBO/rec_turb_eFs_red.F
N3SNATUR/src/noyau90/TURBO/rec_turb_eFs_res.F
N3SNATUR/src/noyau90/TURBO/rec_turb_eFs_tps.F
N3SNATUR/src/noyau90/TURBO/rec_turb_sFe_dim.F
N3SNATUR/src/noyau90/TURBO/rec_turb_sFe_maj.F
N3SNATUR/src/noyau90/TURBO/rec_turb_sFe_red.F
N3SNATUR/src/noyau90/TURBO/rec_turb_sFe_res.F
N3SNATUR/src/noyau90/TURBO/rec_turb_sFe_tps.F
N3SNATUR/src/noyau90/TURBO/send_turb_e2s_dim.F
N3SNATUR/src/noyau90/TURBO/send_turb_e2s_maj.F
N3SNATUR/src/noyau90/TURBO/send_turb_e2s_red.F
N3SNATUR/src/noyau90/TURBO/send_turb_e2s_res.F
N3SNATUR/src/noyau90/TURBO/send_turb_e2s_tps.F
N3SNATUR/src/noyau90/TURBO/send_turb_s2e_ini.F
N3SNATUR/src/noyau90/TURBO/send_turb_s2e_maj.F
N3SNATUR/src/noyau90/TURBO/send_turb_s2e_red.F
N3SNATUR/src/noyau90/TURBO/send_turb_s2e_res.F
N3SNATUR/src/noyau90/TURBO/send_turb_s2e_tps.F
N3SNATUR/src/noyau90/s90/n3snatur_s.F

TAB. B.3 – Routines contenant une référence au communicateur MPI_COMM_WORLD

La compilation

Le script de compilation des sources N3SNatur

La compilation des sources de N3SNatur se fait par le lancement d'un script (TAB. B.4) dans le dossier N3SNATUR/. Il a été rédigé d'après une documentation fournie par SIMULOG.

```
echo "======================"
echo "Compiling noyau ..."
echo "======================"
../bin/SGIMP64/make.gnu vpnaturcMPI

echo "======================"
echo "Compiling NATURC ..."
echo "======================"
../bin/SGIMP64/make.gnu NATURCMPI

echo "======================"
echo "Compiling TURBO ... (Sources non fournies au CERFACS ...)"
echo "======================"
../bin/SGIMP64/make.gnu TURBO

echo "======================"
echo "Compiling noyau90 ..."
echo "======================"
cd src/noyau90
./faire.sh MPI
```

TAB. B.4 – Script de compilation des Sources N3SNatur

Ce script aboutit à la création de librairies et d'exécutables pour l'utilisation de N3SNatur : l'exécutable du code, des outils de découpe de maillage et de solutions, ainsi que de reconstruction de solutions ...

L'option de compilation *PALM*

Afin de ne garder qu'une seule version des sources, une option de compilation a été introduite. Elle est gérable depuis le script de compilation décrit dans la section ci-dessus. La variable concernée est COMP_CHOICE. Si "PALM" est affecté à COMP_CHOICE alors :

- le script de changement de MPI_COMM_WORLD par PL_COMM_EXEC est lancé
- la clef de compilation -DWITHPALM est activée (script getflag_mpi dans N3SNATUR/bin/)
- les librairies et include de PALM sont linkés lors de la compilation (script getflag_mpi dans N3SNATUR/bin/)

Toutes les modifications sur les sources décrites dans cette annexe sont "encadrées" par une clef de compilation. Une étape de précompilation est alors nécessaire pour retenir le code à compiler (FIG B.2).

Code original

```
#ifdef WITHPALM
        CALL PALM_ABORT (info)
#else /* WITHPALM */
        CALL MPI_ABORT (info)
#endif /* WITHPALM */
```

Clef -DWITHPALM activée	Clef -DWITHPALM non activée
CALL PALM_ABORT (info)	CALL MPI_ABORT (info)

FIG. B.2 – Exemple d'utilisation d'une clef de compilation

Annexe C

Le krigeage : une méthode optimale d'interpolation spatiale

Le Krigeage est une méthode optimale, au sens statistique du terme, d'estimation. On peut l'utiliser autant pour l'interpolation que l'extrapolation. Dans cette annexe, nous nous restreindrons à l'interpolation et l'extrapolation spatiales en deux dimensions. Le Krigeage porte le nom de son précurseur, l'ingénieur minier sud-africain D.G. Krige. Dans les années 50, Krige a développé une série de méthodes statistiques empiriques afin de déterminer la distribution spatiale de minerais à partir d'un ensemble de forages. C'est cependant le français Matheron qui a formalisé l'approche en utilisant les corrélations entre les forages pour en estimer la répartition spatiale. C'est lui qui a baptisé la méthode "Krigeage". Il a aussi été le premier à utiliser le terme "géostatistique" pour désigner la modélisation statistique de données spatiales. Les mêmes idées ont été développées parallèlement en URSS par L.S. Gandin. Gandin a baptisé sa méthode "interpolation optimale". Il a introduit la notion d' "analyse objective" pour décrire cette approche basée sur les corrélations. C'est le nom sous lequel la méthode est connue en météorologie. En océanologie, la méthode a été introduite par Bretherton *et al.* et elle est connue sous le nom de "méthode d'interpolation de Gauss-Markov", d'après le nom qu'on lui donne formellement dans les livres de statistiques.

Le texte qui suit est extrait du rapport de projet de fin d'étude de B. Labeyrie [12] (CERFACS, 2004). Le programme de krigeage utilisé lui est aussi dû. Il s'agit d'un code écrit en Fortran 90 utilisant la librairie d'algèbre linéaire LAPACK.

L'interpolation spatiale

L'interpolation spatiale est un problème classique d'estimation d'une fonction $F(\vec{x})$, où $\vec{x} = (x, y)$, en un point quelconque x_p du plan à partir de valeurs connues de F en un certain nombre, n, de points environnants x_i :

$$F(x_p) = \sum_{i=1}^{n} W_i \; F(x_i) \tag{C.1}$$

L'interpolation consiste à déterminer les coefficients de pondération, W_i, de chacun des points environnants. Les deux méthodes les plus connues sont l'interpolation linéaire (en fonction de l'inverse de la distance) et la méthode des splines cubiques (ajustement de polynômes cubiques).

Description du krigeage

Le krigeage définit les coefficients de pondération à partir de la covariance entre les points en fonction de la distance entre ces points. Les estimateurs de krigeage sont statistiquement non biaisés (la valeur prédite et la valeur réelle coïncident en moyenne) et permettent de minimiser la variance de l'erreur sur les valeurs prédites tout en donnant une estimation.

La principale hypothèse nécessaire pour utiliser cette méthode est la stationnarité de la moyenne et de la variance de la fonction à interpoler, c'est à dire que la moyenne et la variance ne doivent pas dépendre de la position des points mais uniquement de la distance entre eux.

Les différents types de krigeage

Il existe trois types de krigeage à une seule variable. La différence entre ces types réside dans la connaissance de la variable à interpoler.

- le krigeage simple : la variable à interpoler est stationnaire et de moyenne connue.
- le krigeage ordinaire : la variable à interpoler est stationnaire et de moyenne inconnue, c'est la méthode la plus utilisée.
- le krigeage universel : la variable à interpoler est non-stationnaire mais présente une tendance.

Théorie du krigeage ordinaire

Soit $\hat{f}$ l'estimation de la fonction à interpoler. On a par définition :

$$\hat{f}(x) = \sum_{i=1}^{n} W_i(x)\ F(x_i) \tag{C.2}$$

La moyenne et la variance étant stationnaires, on a :

$$C[f(x_i), f(x_j)] = C(|x_i - x_j|) \tag{C.3}$$

avec C la fonction covariance définie par :

$$C(x_1, x_2) = E\left[(f(x_1) - \mu)\,(f(x_2) - \mu)\right] \quad \text{avec } E \text{ l'espérance mathématique et } \mu \text{ la moyenne} \tag{C.4}$$

On peut donc définir une matrice de covariance $\underline{C}$:

$$\underline{C} = \begin{pmatrix} \sigma^2 & C(|x_1 - x_2|) & \cdots & C(|x_1 - x_n|) \\ C(|x_2 - x_1|) & \sigma^2 & \cdots & C(|x_2 - x_n|) \\ \vdots & \vdots & \ddots & \vdots \\ C(|x_n - x_1|) & C(|x_n - x_2|) & \cdots & \sigma^2 = C(0) \end{pmatrix} \tag{C.5}$$

et un vecteur de covariance $\vec{c}$:

$$\vec{c}(x) = \begin{pmatrix} C(|x - x_1|) \\ C(|x - x_2|) \\ \vdots \\ C(|x - x_n|) \end{pmatrix} \tag{C.6}$$

Pour que l'estimateur de krigeage soit optimum, il faut que la variance de l'erreur MSE (Mean Square Error) soit minimale :

$$\begin{aligned} MSE\ \hat{f}(x) &= E\,[\hat{f}(x) - f(x)]^2 && \text{(C.7)} \\ &= E\,[\hat{f}(x)]^2 + E\,[f(x)]^2 - 2\ E\,[f(x)\hat{f}(x)]^2 && \text{(C.8)} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} w_j(x)\,C(|x_i - x_j|) + \sigma^2 - 2\sum_{i=1}^{n} C(|x - x_i|) && \text{(C.9)} \\ &= W^T(x)\ C\ W(x) + \sigma^2 - 2\ W^T(x)\ \vec{c}(x) && \text{(C.10)} \end{aligned}$$

On peut donc trouver une solution à ce problème de minimisation en différenciant la variance de l'erreur de $f(x)$ par $W(x)$:

$$\frac{\partial}{\partial W(x)}\ MSE\ \hat{f}(x) = 0 \qquad \text{(C.11)}$$

ce qui nous donne :

$$W(x) = \underline{C}^{-1}\ \vec{c}(x) \qquad \text{(C.12)}$$

De plus, la condition de solution non biaisée impose que la somme des coefficients de pondération soit égale à 1.

$$\sum_{i=1}^{n} W_i(x) = 1 \qquad \text{(C.13)}$$

Le système à résoudre est donc modifié en :

$$W_+(x) = \underline{C}_+^{-1}\ \vec{c}_+(x) \qquad \text{(C.14)}$$

avec

$$w_+(x) = \begin{pmatrix} w_1(x) \\ w_2(x) \\ \vdots \\ w_n(x) \\ \lambda(x) \end{pmatrix} \qquad \text{(C.15)}$$

$$\underline{C}_+ = \begin{pmatrix} \sigma^2 & C(|x_1 - x_2|) & \cdots & C(|x_1 - x_n|) & 1 \\ C(|x_2 - x_1|) & \sigma^2 & \cdots & C(|x_2 - x_n|) & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ C(|x_n - x_1|) & C(|x_n - x_2|) & \cdots & \sigma^2 = C(0) & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \qquad \text{(C.16)}$$

et

$$\vec{c}_+(x) = \begin{pmatrix} C(|x - x_1|) \\ C(|x - x_2|) \\ \vdots \\ C(|x - x_n|) \\ 1 \end{pmatrix} \qquad \text{(C.17)}$$

où $\lambda(x)$ est un nouvelle fonction appelée multiplieur de Lagrange.

Pour le krigeage ordinaire, on peut se donner arbitrairement un covariogramme $C(h)$ à partir de fonctions théoriques typiques ou bien l'estimer à partir de valeurs observées en des points connus par la méthode des moindres carrés.

Ici on prend :

$$C(h) = 20\,\exp\,(-\frac{3\,h}{100}) \qquad \text{(C.18)}$$

Exemple d'utilisation de la méthode

Le programme utilisé repose sur la théorie du krigeage ordinaire décrite dans la section précédente. Il utilise un fichier d'entrée pour la définition des paramètres de l'interpolation et des fichiers de données contenant les valeurs des variables à interpoler pour un nuage de points.

Pour le fichier d'entrée, il faut renseigner :
- le nombre de variables à interpoler
- le nombre de fichiers de données pour chaque variable et leurs noms
- si on veut combiner deux variables et dans ce cas choisir un type de combinaison (combinaison de type rapport des variables ou somme pondérée des variables)
- le nombre de points en abscisse et en ordonnée désirés pour l'interpolation

Les fichiers de données précisent pour chaque point son abscisse, son ordonnée et la valeur de la fonction en ce point.

En vue de tester ce code et de le prendre en main, un essai sur la fonction suivante a été réalisé :

$$\begin{aligned} &f(x,y) = 1 - cos(x)cos\left(\tfrac{y}{\sqrt{2}}\right) + \tfrac{1}{50}\left[\left(x - \tfrac{1}{2}y\right)^2 + 1.75y^2\right] \\ &(x,y) \in [-10,10]^2 \end{aligned} \tag{C.19}$$

Un programme écrit en Fortran permet de générer des fichiers d'entrée du code de krigeage :
- un fichier contenant 100 points sur une grille de pas Δx et Δy fixés
- deux fichiers contenant 100 points pris aléatoirements dans le domaine de définitions de f

Le but est de comparer les résultats donnés par les fichiers de points générés aléatoirement avec la référence qui est le fichier de points pris sur une grille. La grille de krigeage est de 100×100 points.

La figure C.1 montre pour chaque fichier de données une représentation de la courbe en 3D puis en 2D ainsi que le tracé de la variance de l'erreur. On y constate que globalement, la technique de krigeage donne une bonne approximation de la tendance de la courbe dans le cas des fichiers de points aléatoires. Notamment, les minimums retrouvés sont approximativement les mêmes que la référence. Concernant les maximums, on peut remarquer un écart important, (autour de 9 pour la référence et aléa 2 contre 8.5 pour aléa 1) : ceci s'explique par le fait qu'ils se situent sur les bords du domaine d'étude qui sont mal représentés par le tirage aléatoire de points. Les graphiques de variance renseignent sur les zones de certitude ou d'incertitude concernant la connaissance de la fonction. Ils permettent aussi de localiser les points qui ont servis au krigeage : naturellement, ils sont le lieu de variance d'erreur minimum.

La méthode de krigeage est donc très performante pour représenter une courbe qui passe par un nuage de point. Elle sera très utile pour comprendre le comportement du simplexe lors de l'optimisation. Dans cette situation, il faudra toujours garder en tête qu'il y a des zones non explorées par le simplexe dans lequelles il peut y avoir des minimums qui ne sont pas vu par l'algorithme de krigeage. On prendra garde notamment aux bords des domaines.

La méthode de krigeage pourra donc servir à construire (soit au fur et à mesure des runs d'optimisation soit par un balayage grossier du domaine d'étude) des surfaces de réponse dans le but de placer le simplexe de manière optimale à l'initialisation dans un bassin d'attraction. Remarquons que le code a été développé pour uniquement deux paramètres. Par la suite, il faudra donc constater les possibilités d'étendre la méthode à un plus grand nombre de paramètres.

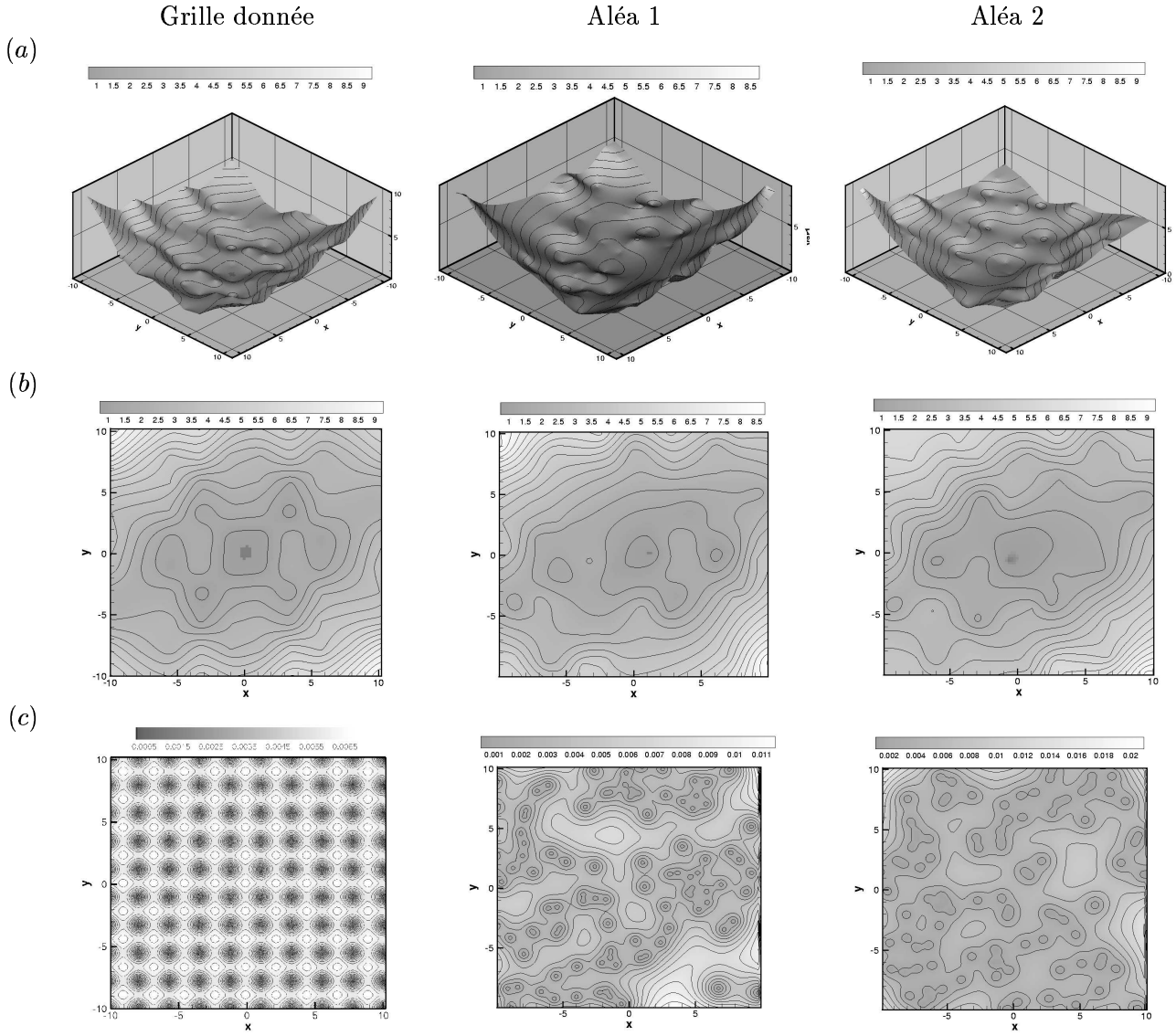


FIG. C.1 – Fonction test : vue 3D de la fonction (a), vue 2D de la fonction, variance de l'erreur de la méthode de krigeage (c)

Application de la méthode de krigeage dans un cas simple

Cette section présente sur un cas simple une utilisation possible du code de krigeage dans le cadre des travaux d'optimisation. Elle permet en outre de présenter l'outil qui permet de faire le mapping de la fonction objectif, construit à partir de PALM et de l'unité N3SNatur.

Cas étudié : canal 2D

Le cas étudié est un canal 2D de dimensions $0.6 \times 0.02m$ comportant 441 noeuds et 800 éléments. Les variables d'optimisation sont les vitesses longitudinale U_e et transversales V_e à l'entrée du canal (optimisation 2D). Ces vitesses sont données constantes sur la condition à la limite d'entrée. La fonction objectif est basée sur un profil de vitesse U_{ref} en sortie du canal obtenu par un run N3SNatur préalable (U_e et V_e fixés : $(U_e, V_e) = (38.8, -2.19)ms^{-1}$) et le profil de vitesse de l'état calculé $U(U_e, V_e)$:

$$f(U_e, V_e) = \alpha \left[1 - \frac{U(U_e,V_e)}{U_{ref}}\right]^2$$
$$(U_e, V_e) \in [20, 60] \times [-5, 10](ms^{-1}) \tag{C.20}$$

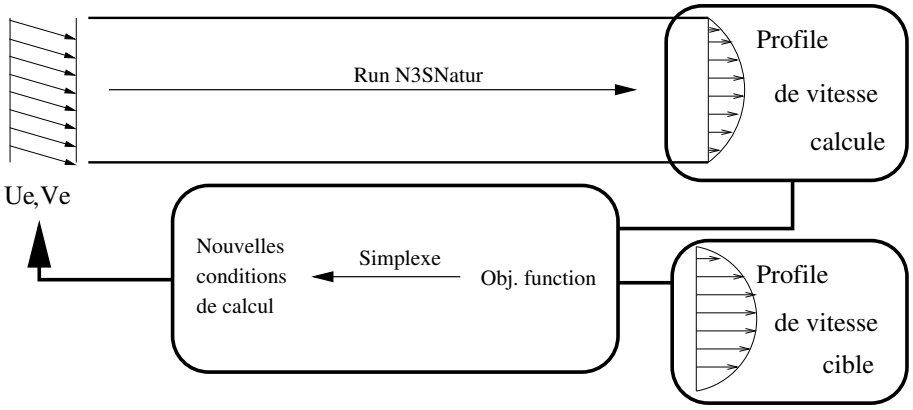


FIG. C.2 – Configuration d'étude pour le canal 2D

Mapping de la fonction objectif

Afin de placer convenablement le simplexe à l'initialisation, un mapping de la fonction objectif est réalisé : la fonction objectif est calculée sur les points d'une grille 11×16 (*i.e.* entre 20 et 60 ms^{-1} pour U_e, entre -5 et 10 ms^{-1} pour V_e). Pour cela, l'unité N3SNatur est insérée dans une boucle créé dans PrePALM (FIG. C.3), qui lui fournit les points de calcul. Puis, le code de krigeage est utilisé avec en entrée les points calculés dans la boucle. Le krigeage est alors fait sur une grille de 100×100 points dont la représentation est donnée en 2D et en 3D sur les figures C.4. On y constate deux minimums : (38.8,-2.19) qui est global et (38.8,10) qui est local sur l'espace d'état.

Résultats de l'optimisation

Deux runs d'optimisation ont été effectués :
- un premier pour lequel le simplexe est initialisé dans le bassin d'attraction du minimum local (38.8,10) : paramètre lié à V_e positif
- un second avec le simplexe initialisé dans le bassin d'attraction du minimum global recherché (38.8,-2.19)

Dans chacun des cas, l'algorithme converge ($\epsilon = 10^{-6}$) vers le minimum du bassin d'attraction comme le montrent les figures C.5 et C.6 qui représentent l'évolution du simplexe aucours des itérations. Notons que sur ces figures, "1rst parameter" correspond à la vitesse Ue et "2nd parameter" à la vitesse Ve.

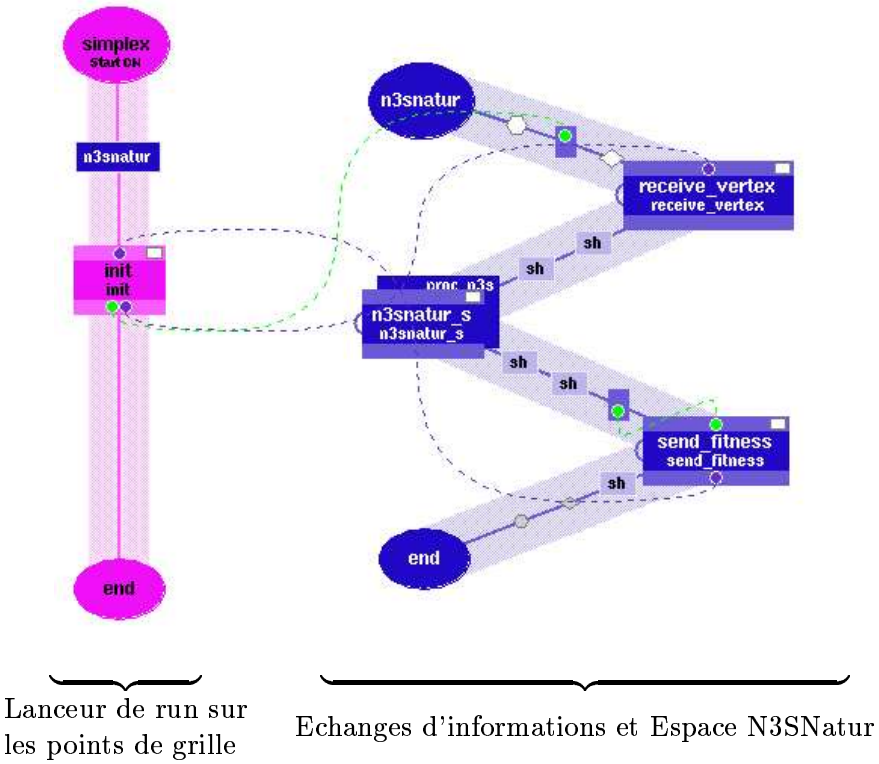


FIG. C.3 – Boucle de calcul pour le mapping de fonction objectif sous PrePALM

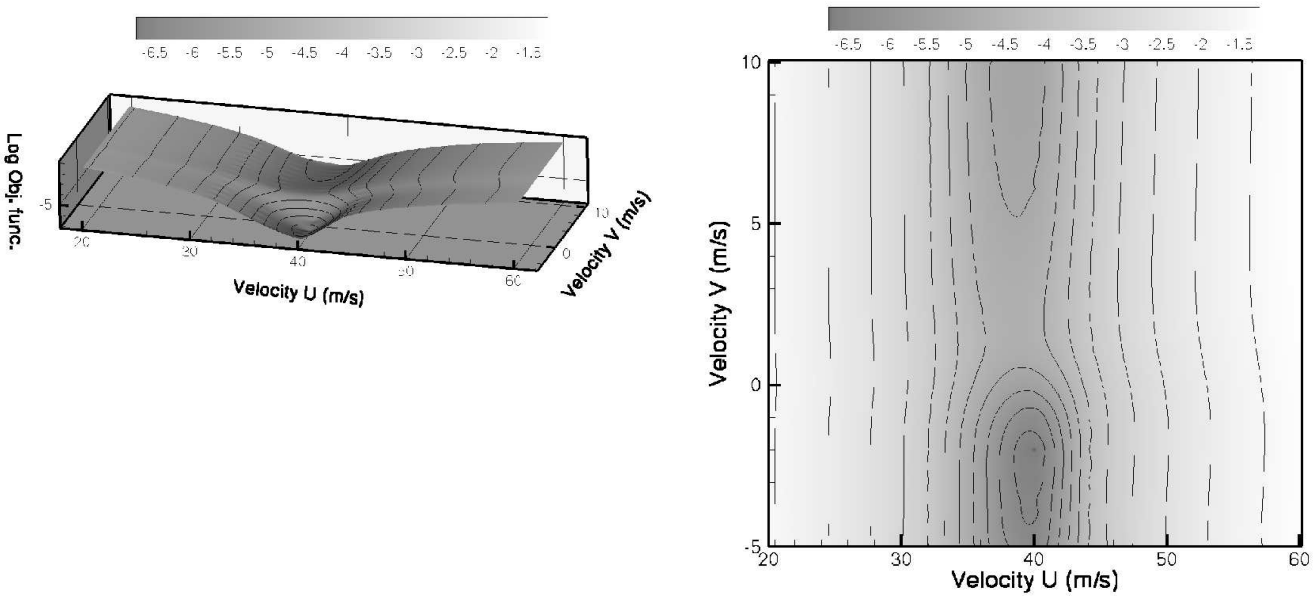


FIG. C.4 – Mapping de la fonction objectif dans le cas du canal 2D avec deux paramètres d'optimisation

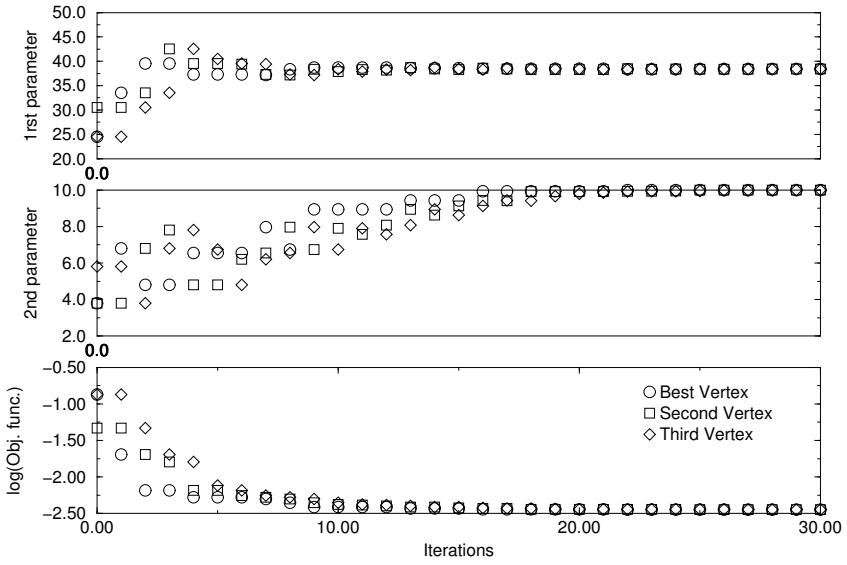


FIG. C.5 – Evolution du simplexe et du logarithme de la fonction objectif dans le cadre de l'optimisation sur un canal 2D avec deux paramètres d'optimisation : initialisation dans le bassin d'attaction du minimum local (38.8,10)

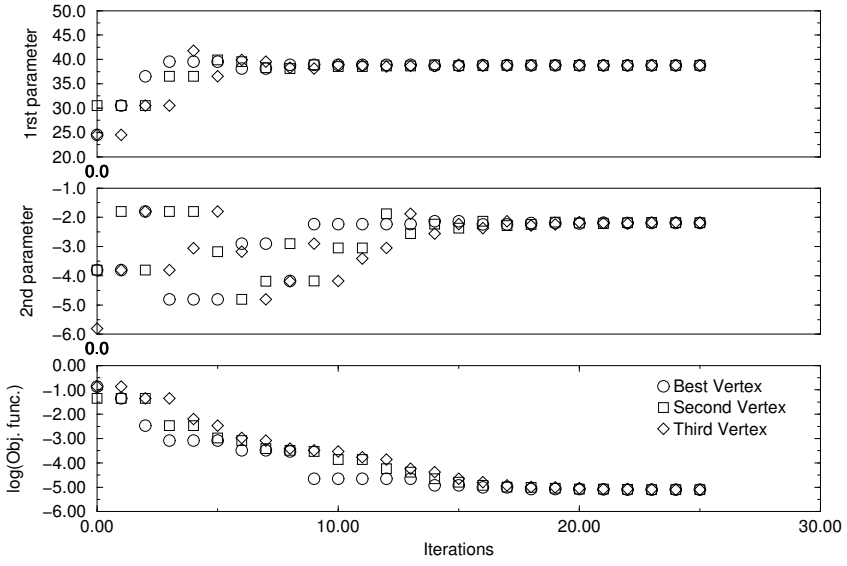


FIG. C.6 – Evolution du simplexe et du logarithme de la fonction objectif dans le cadre de l'optimisation sur un canal 2D avec deux paramètres d'optimisation : initialisation dans le bassin d'attaction du minimum global (38.8,-2.19)

Annexe D

Précisions sur les paramètres de calcul N3SNatur des runs d'optimisation

Cette annexe présente tout d'abord les paramètres utilisés pour les calculs réalisés avec N3SNatur dans le cadre des runs d'optimisation. Ensuite, les calculs préliminaires dont il est question dans le chapitre 4 sont vu plus en détail.

Les paramètres de calcul N3SNatur

Paramètres de maillages :

Nombre de dimension spaciale :	**2**
Type d'élément :	**triangles**
Nombre total de noeuds :	**5730**
Nombre total d'éléments :	**11016**
Longueur :	$L \times l = 0.5\ \mathbf{m} \times 0.06\ \mathbf{m}$

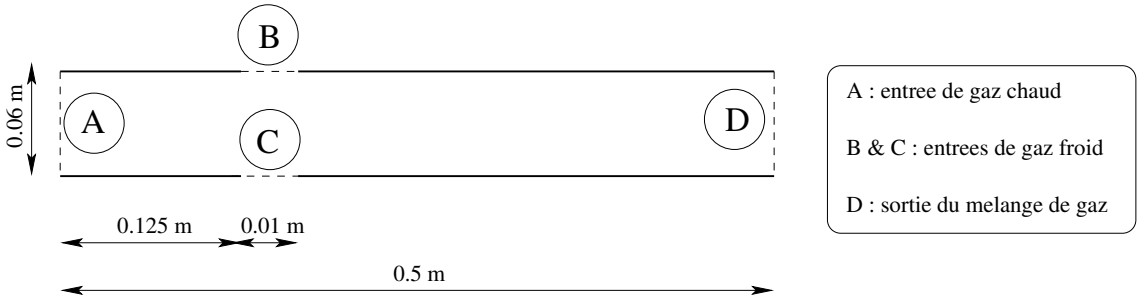


FIG. D.1 – Géométrie utilisée pour les calculs N3SNatur

Propriétés générales du fluide :

Equations résolues	**Navier-Stokes**
Régime d'écoulement :	**Turbulent**
Modèle de turbulence :	$k-\epsilon$ **standard, Loi LTM pour les parois**
Modèle de chimie :	**Pas de chimie**
Nombre d'espèce(s) :	**1**

Conditions initiales et aux limites

Conditions initiales :	$U = U_A = 30$ **m/s**
	$V = 0$ **m/s**
	$P = 101325$ **Pa**
	$T = T_A = 1500$ **K**
	$k = 3$ $\mathbf{m^2/s^2}$
	$\varepsilon = 600$ $\mathbf{m^2/s^3}$
Conditions aux limites :	
Entrée A	
Débit température	$U = 30$ **m/s**
	$V = 0$ **m/s**
	$P = 101325$ **Pa**
	$T = 1500$ **K**
	$k = 3$ $\mathbf{m^2/s^2}$
	$\varepsilon = 600$ $\mathbf{m^2/s^3}$
Entrée B	
Débit température	$U = 0$ **m/s**
	$V = V_B$ **m/s**
	$P = 101325$ **Pa**
	$T = T_B$ **K**
	$k = 3$ $\mathbf{m^2/s^2}$
	$\varepsilon = 600$ $\mathbf{m^2/s^3}$
Entrée C	
Débit température	$U = 0$ **m/s**
	$V = V_C$ **m/s**
	$P = 101325$ **Pa**
	$T = T_C$ **K**
	$k = 3$ $\mathbf{m^2/s^2}$
	$\varepsilon = 600$ $\mathbf{m^2/s^3}$
Sortie D	
Sortie à pression moyenne imposée	$P = 101325$ **Pa**
Parois	
Loi de paroi LTM	$U = 0$ **m/s**
	$V = 0$ **m/s**
	$T = 300$ **K**

Paramètres numériques :

Généralités	
Type de schéma Euler pour MUSCL :	**Implicite Gauss-Seidel**
Schéma temporel	
Interpolation temporelle pour Euler :	**Euler ordre 1**
Méthode implicite	
Type de schéma visqueux pour MUSCL :	**Implicite**
Type de schéma source pour $k-\varepsilon$ pour MUSCL :	**Implicite**
Type de schéma pour les T.S. de rotation	**Implicite**
Nombre maxi de relaxation :	**200**
Critère d'arrêt Gauss Seidel-Jacobi :	10^{-3}
Optimisation Gauss Seidel :	**Optimisation selon x**
Turbulence	
Nombre max de relaxation $(K-\epsilon)$:	**100**
Résidu min :	10^{-6}
Schéma spacial	
Choix du flux numérique Euler pour MUSCL :	**ROE TOUMI**
Traitement HLLE :	**HLLE active**
Traitement Roe-Turkel pour le bas Mach :	**ROE - TURKEL desactive**
Interpolation spaciale de la convection :	**Ordre 2 avec limiteur de Van Albada**
Paramètres de calcul	
Pas de temps :	**constant, calculé par N3SNatur**
Nombre de régions parallèles	**5**
Coefficient du nombre de Courant	**20**
Nombre de Courant max	**5**

Résultats des calculs préliminaires

Applications analytiques

Afin de se donner une idée des températures que l'on peut obtenir en sortie du dispositif présenté sur la figure D.1 en fonction des conditions d'entrée, il est nécessaire de procéder à des calculs analytiques dont les hypothèses sont les suivantes :

- les profils des variables Navier Stokes sont constants sur les conditions aux limites d'entrées et de sortie
- les chaleurs spécifiques Cp du gaz sont indépandantes de la température
- le gaz est parfait : $\frac{P}{\rho} = rT$

Le bilan enthalpique cas test est donné par :

$$\sum_{i=A}^{C} \dot{m}_i \int_{T_O}^{T_i} Cp^i dT = \dot{m}_D \int_{T_O}^{T_D} Cp^D dT \tag{D.1}$$

avec $\dot{m}_i$ (Kgs^{-1}) le débit, Cp^i $(JKg^{-1}K^{-1})$ la chaleur spécifique, T_i (K) la température sur la condition à la limite i et T_0 (K) une température de référence (souvent $T_0 = 0K$ ou $T_0 = 298.15K$).

En considérant les hypothèses exposées, le bilan enthalpique D.1 et la conservation de la masse :

$$\sum_{i=A}^{C} \dot{m}_i = \dot{m}_D \tag{D.2}$$

on obtient l'équation suivante qui permet d'obtenir T_D :

$$\left(\frac{S_A U_A}{T_A} + \frac{S_B V_B}{T_B} + \frac{S_C V_C}{T_C}\right) T_D = S_A U_A + S_B V_B + S_C V_C \tag{D.3}$$

avec S_i (m^2) la section de la condition à la limite i et (U_i, V_i) (ms^{-1}) les composantes de la vitesse à la condition à la limite i.

Le tableau D.1 donne les températures de sortie dans des cas qui serviront de références pour valider les calculs effectués avec le code N3SNatur. Notons dès à présent que les valeurs étudiées ici serviront de bornes aux variables d'optimisation qui seront T_B, T_C, V_B et V_C. Les grandeurs suivantes sont constantes :

- $S_A = 0.06\ m^2$
- $S_B = S_C = 0.01\ m^2$
- $U_A = 30\ ms^{-1}$
- $T_A = 1500\ K$

Cas	$T_B = T_C = 300\ K$	$T_B = T_C = 500\ K$	Run N3SNatur ($T_B = T_C = 300\ K$)
$V_B = V_C = 40\ ms^{-1}$	672 K	930 K	670 K
$V_B = V_C = 100\ ms^{-1}$	483 K	730 K	487 K
$V_B = V_C = 150\ ms^{-1}$	428 K	670 K	439 K

TAB. D.1 – Calculs analytiques de températures de sorties en fonction des conditions d'injection

Applications numériques

Des calculs avec le code N3SNatur ont été réalisés afin d'optimiser au maximum le temps de restitution du code pour le calcul du champs fluide. C'est en effet cette partie qui coûte le plus cher dans la boucle d'optimisation. Le but était de trouver le nombre d'itérations minimum à faire pour atteindre une convergence satisfaisante du calcul. Les trois calculs présentés dans cette annexe sont l'aboutissement d'une campagne d'essais réalisée à cet effet. Ils correspondent aux valeurs limites que l'on se fixera en terme de vitesse V sur les frontières B et C.

La différence entre ces calculs est la valeur des vitesses $V_B = -V_C$:

- $V_B = V_C = 40\ ms^{-1}$ et $T_B = T_C = 300\ K$
- $V_B = V_C = 100\ ms^{-1}$ et $T_B = T_C = 300\ K$
- $V_B = V_C = 150\ ms^{-1}$ et $T_B = T_C = 300\ K$

Le nombre d'itération est fixé à 6000. La figure D.2 montre l'évolution du résidu de ρ pour les trois calculs. La figure D.3 représente l'évolution des débits entrant ($\sum_{i=A}^{C} \dot{m}_i$) et sortant ($\dot{m}_D$) du domaine d'une part, et l'évolution de la température au milieu de la condition à la limite de sortie D en fonction des itérations d'autre part.

L'égalité entre les débits entrant et sortant ainsi que la stabilisation de la température de sortie sont des critères de convergence assez fiable pour le cas d'optimisation que l'on souhaite traiter. Attendre une convergence au sens définie dans le code, c'est à dire basée sur les résidus, serait trop fastidieuse.

On constate tout d'abord que le code N3SNatur donne pour la valeur moyenne de T_D sur la condition à la limite de sortie, avec ces conditions de run, des valeurs très proches de celles

trouvées analytiquement (TAB. D.1). Ensuite, le cas le plus défavorbale dans de ces tests en nombre d'itérations est celui où $V_B = -V_C = -40ms^{-1}$. Le nombre d'itérations minimum pour s'assurer ce type de convergence est alors de l'ordre de 3000. Donc les runs N3SNatur effectués dans le cadre de l'optimisation le seront sur 3000 itérations.

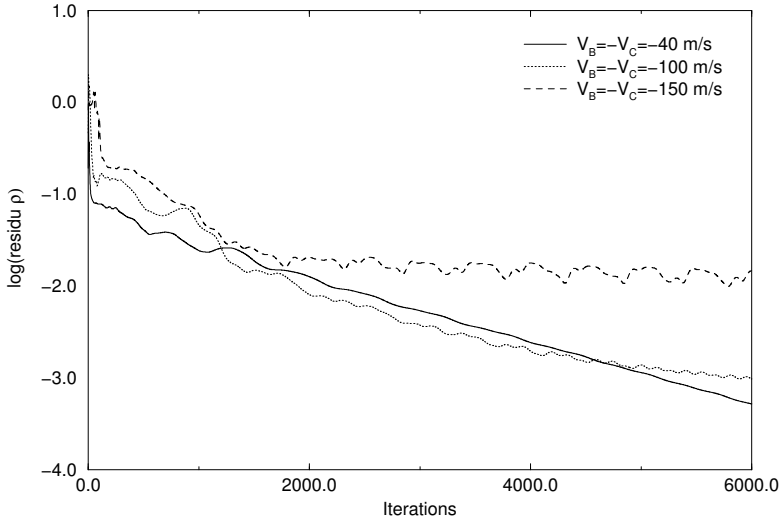


FIG. D.2 – Evolution des résidu de ρ en fonction des itérations pour les trois calculs

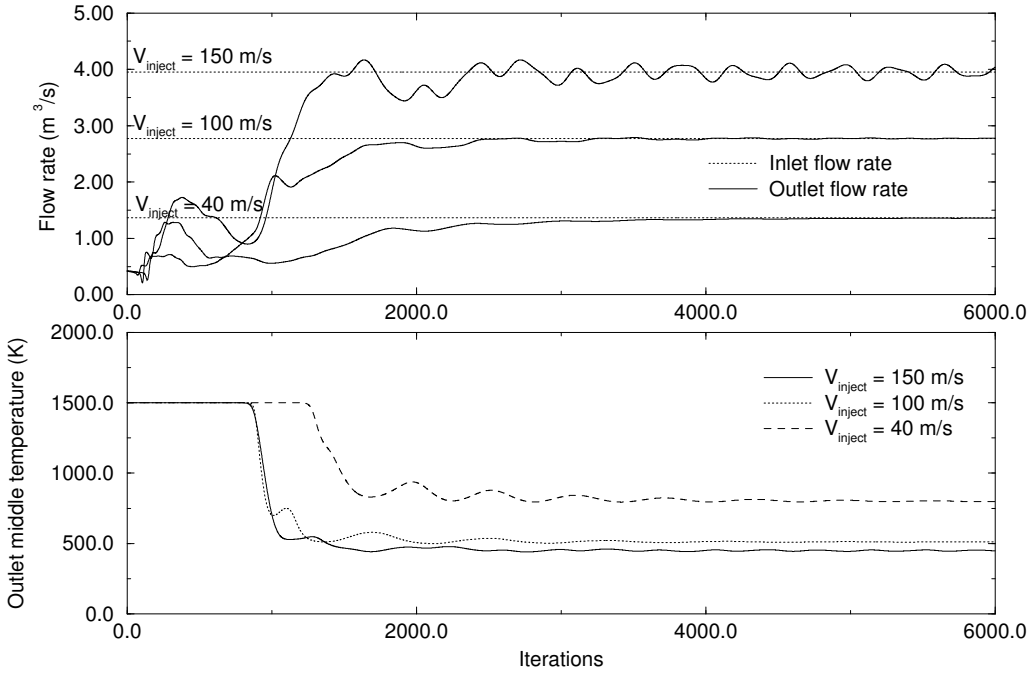


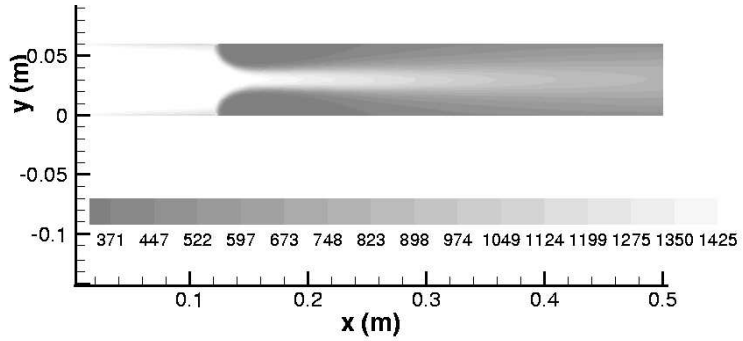
FIG. D.3 – Evolution des débits entrant et sortant en fonction et de la température de sortie au cours des itérations pour les trois calculs

Le tableau D.2 donne pour chacun des calculs les temps de restitution pour faire 6000 itérations et les temps physiques simulés. (dans ces essais, le domaine était divisé en 7 régions alors que dans le cas de l'optimisation, il y a 5 régions)

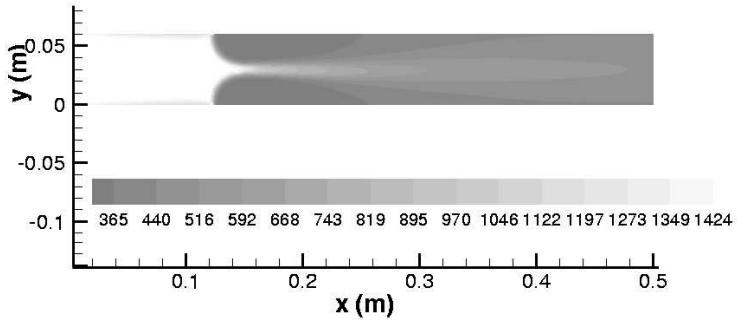
Les figures D.4 montrent les champs de températures obtenus à 6000 itérations dans chaque cas.

Cas	Temps (s)	Temps physique (s)
$V_B = V_C = 40\ ms^{-1}$	4877.68	0.0377
$V_B = V_C = 100\ ms^{-1}$	4738.14	0.0337
$V_B = V_C = 150\ ms^{-1}$	5088.71	0.0258

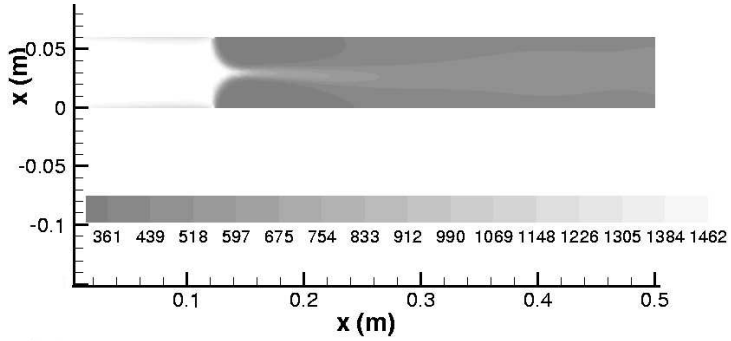
TAB. D.2 – Indications sur les temps de calculs



$V_B = -V_C = -40\ ms^{-1}$



$V_B = -V_C = -100\ ms^{-1}$



$V_B = -V_C = -150\ ms^{-1}$

FIG. D.4 – Champs de température pour les cas présentés